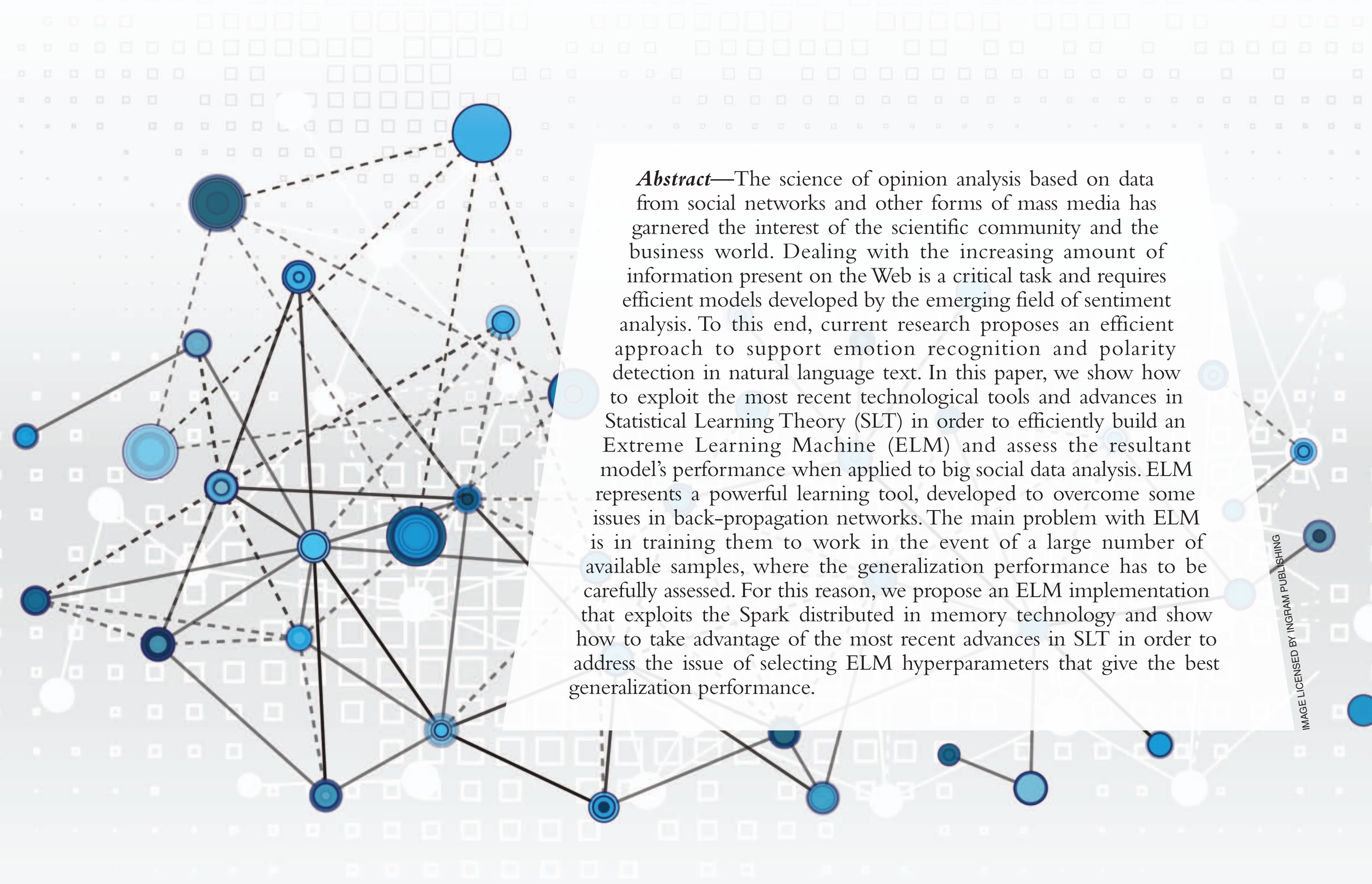

Abstract—The science of opinion analysis based on data from social networks and other forms of mass media has garnered the interest of the scientific community and the business world. Dealing with the increasing amount of information present on the Web is a critical task and requires efficient models developed by the emerging field of sentiment analysis. To this end, current research proposes an efficient approach to support emotion recognition and polarity detection in natural language text. In this paper, we show how to exploit the most recent technological tools and advances in Statistical Learning Theory (SLT) in order to efficiently build an Extreme Learning Machine (ELM) and assess the resultant model's performance when applied to big social data analysis. ELM represents a powerful learning tool, developed to overcome some issues in back-propagation networks. The main problem with ELM is in training them to work in the event of a large number of available samples, where the generalization performance has to be carefully assessed. For this reason, we propose an ELM implementation that exploits the Spark distributed in memory technology and show how to take advantage of the most recent advances in SLT in order to address the issue of selecting ELM hyperparameters that give the best generalization performance.

Statistical Learning Theory and ELM for Big Social Data Analysis

Luca Oneto
DIBRIS, University of Genoa, ITALY

Federica Bisio
DITEN, University of Genoa, ITALY

Erik Cambria
School of Computer Science and Engineering, Nanyang Technological University, SINGAPORE

Davide Anguita
DIBRIS, University of Genoa, ITALY

I. Introduction

The information age has brought along an explosion of Big Data [1], from multiple sources in every aspect of our lives: human activity signals from wearable sensors and personal devices, experiments in particle discovery research and stock market data systems are few examples. Big social data analysis [2] is the area of research focusing on collecting, examining and processing large multi-modal and multi-source datasets in order to discover patterns/correlations and extract information from the Social Web. This is usually accomplished through the use of computationally expensive supervised and unsupervised machine learning algorithms that learn from the available data (e.g., Support Vector Machines-SVMs [3], Artificial Neural Networks-ANNs, [4], *k*-Nearest Neighbors-kNN [5], and Random Forests-RF [6]) that are not able to

Digital Object Identifier 10.1109/MCI.2016.2572540
Date of publication: 18 July 2016

Corresponding Author: Erik Cambria (Email: cambria@ntu.edu.sg).

handle current data volumes [7]. Parallel approaches have been proposed in order to boost processing speeds [8] but this clearly requires technologies that support distributed computations.

Extreme learning machine (ELM) [9] is an emerging learning paradigm, presenting an efficient unified solution to generalized feed-forward neural networks. Unlike ANNs, however, ELM cannot be easily parallelized, due to the presence of a pseudo-inverse calculation [10]. Therefore, this paper aims to find a reliable method to realize a parallel implementation of ELM that can be applied to large datasets typical of Big Data problems. An example of parallel ELM implementation for regression based on the MapReduce framework can be found in [11], while [12] provides a parallel ensemble method for an Online Sequential ELM variant.

Several technologies that exploit multiple levels of parallelism (e.g., multi-core, many-core, GPU, cluster, etc.) are currently available [13]–[16]. Spark [17] in combination with cloud computing [18], [19] is a state-of-the-art framework for high performance parallel computing designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data, such as supervised machine learning algorithms.

Apart from building supervised learning models efficiently and with scalable algorithms, another important issue in Big Data is how to effectively and efficiently assess the performance of a predictive model. Data-driven models exploit non-parametric inference, where it is expected that an effective model would stem directly from the data, without any assumption on the model family nor any other information that is external to the dataset itself [20]. With the advent of the Big Data era, this approach has increasingly gained popularity, with the belief that effective predictive models, with the desired accuracy, can be generated by simply collecting larger volumes of data (see, as an example, [21] for some insights on this provocative and inexact but, unfortunately, widespread belief).

Statistical Learning Theory (SLT) addresses the problem of assessing the performance of a predictive model, by trying to find necessary and sufficient conditions for non-parametric inference to build predictive models from data [22]–[26] or, in the language of SLT, learn an optimal model from data. For a long time, SLT was considered only a theoretical, albeit very sound and deep, statistical framework, without any real applicability to practical problems [27]. With important advances in this field over the last decade [28], it has been shown that SLT can provide practical answers, at least when targeting the inference of data-driven models for classification purposes [29], [30].

II. Related Work

In recent years, opinions and sentiments of the masses have increasingly been publicly conveyed through social networks, web communities, blogs, wikis, and other online collaborative media. This has deeply changed the way people share knowledge and communicate experiences. As a result, the distillation of useful information from the massive amount of opinions is a key tool for marketers trying to create a product, brand, or organization image or identity in the minds of their customers. This has led to an in-depth development of the field of sentiment analysis, which deals with information retrieval and knowledge discovery from text using data mining and natural language processing (NLP) techniques [31].

Main approaches to big social data analysis can be broadly grouped into two categories: knowledge-based techniques [32] and statistical methods [33]. While the former mainly leverage on ontologies [34], lexicons [35], semantic networks [36], or patterns [37], the latter are gradually shifting to the adoption of ELM, deep learning and convolutional neural network (CNN).

In particular, Tang et al. [38] developed a CNN-based approach to obtain word embeddings for words popularly used in tweets, then fed them into the network for sentiment analysis. A deep CNN for sentiment detection in short text was also proposed by Santos et al. [39]. The approach based on Sentiment Specific Word Embeddings [40] considers word embeddings based on a sentiment corpora: this means including more affective clues than regular word vectors, thus producing a better result. The importance of studying the granularity of emotions in social networks is underlined by the new project 'Reactions', developed by Facebook: instead of only the Like button, a more complete choice of emotions is proposed (e.g., 'love', 'fun', 'anger', 'surprise', 'sadness'). A similar approach was adopted by Poria et al. [41], which extracted features from short texts, based on the activation values of an inner layer of a deep CNN, and used these for multimodal sentiment analysis.

III. Preliminary Definitions

Let us first focus on the binary classification problem [4], [22], [42]. Let $\mathcal{X} \in \mathbb{R}^d$ and $\mathcal{Y} \in \{\pm 1\}$ be, respectively, the input and the output spaces. We consider a set of labeled independent and identically distributed (i.i.d.) data $\mathcal{D}_n = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\}$ of size n, where $\boldsymbol{z}_{i \in \{1,\dots,n\}} = (\boldsymbol{x}_i, y_i)$ with $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, sampled from an unknown distribution μ. As we are targeting at Big Data problems [7], [43], [44], we will focus on the case where n is very large. For later use we also define two modified training sets: $\mathcal{D}_n^{\setminus i} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_{i-1}, \boldsymbol{z}_{i+1}, \dots, \boldsymbol{z}_n\}$ and $\mathcal{D}_n^{i} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_{i-1}, \boldsymbol{z}'_i, \boldsymbol{z}_{i+1}, \dots, \boldsymbol{z}_n\}$, where, respectively, the i-th element is removed or replaced by another sample [25].

A learning algorithm $\mathcal{A}_{\mathcal{H}}$, characterized by a set of hyperparameters $\mathcal{H}$ that must be tuned, maps $\mathcal{D}_n$ into a function $f: \mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}$ from $\mathcal{X}$ to $\mathcal{Y}$. In particular, $\mathcal{A}_{\mathcal{H}}$ allows designing $f \in \mathcal{F}_{\mathcal{H}}$ and the class of functions $\mathcal{F}_{\mathcal{H}}$, which is generally unknown (and depends on $\mathcal{H}$) [26], [28], [30]. The accuracy of a function $f: \mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}$ in representing the hidden relationship μ is measured with reference to a loss function $\ell(f, \boldsymbol{z}): \mathcal{F}_{\mathcal{H}} \times (\mathcal{X} \times \mathcal{Y}) \to [0,1]$ [25]. In particular, since we are dealing with binary classification problems, the loss function (called the *hard* loss) simply counts the number of misclassified examples [22], [45]: $\ell_H(f, \boldsymbol{z}) = [y f(\boldsymbol{x}) \le 0] \in \{0, 1\}$.

The quantity which we are interested in is the generalization error [22], [30], namely the error that a model will perform on new data generated by μ and previously unseen $L(f) = \mathbb{E}_{\boldsymbol{z}} \ell(f, \boldsymbol{z})$. Unfortunately, since μ is unknown, $L(f)$ cannot be computed and, consequently, must be estimated.

Two common empirical estimators are the empirical $\hat{L}_{\text{emp}}(f) = 1/n\sum_{z \in \mathcal{D}_n} \ell(f, z)$ [22] and leave-one-out $\hat{L}_{\text{loo}}(f) = 1/n\sum_{z_i \in \{1,\ldots,n\} \in \mathcal{D}_n} \ell(\mathscr{A}_{(\mathcal{D}_n^{\setminus i}, \mathcal{H})}, z_i)$ [46] errors.

IV. Extreme Learning Machines

The ELM approach [9], [47], [48] was introduced to overcome problems posed by the back-propagation training algorithm [49]; specifically, potentially slow convergence rates, the critical tuning of optimization parameters, and the presence of local minima that call for multi-start and re-training strategies. In this section, we recall conventional ELM and then adapt it to the Big Data framework. ELM was originally developed for the single hidden layer feedforward neural networks [50], [51] and then generalized in order to cope with cases where ELM is not neuron alike:

$$f(\boldsymbol{x}) = \sum_{j=1}^{h} \boldsymbol{w}_j g_j(\boldsymbol{x}), \tag{1}$$

where $g_j: \mathbb{R}^d \to \mathbb{R}, j \in \{1, \ldots, h\}$ is the hidden layer output corresponding to the input sample $\boldsymbol{x}$ and $\boldsymbol{w}$ is the output weight vector between the hidden layer and the output layer.

In our case, the input layer has d neurons and connects to the hidden layer (having h neurons) through a set of weights

$$\boldsymbol{v}_j \in \mathbb{R}^d, \quad j \in \{1, \ldots, h\}, \tag{2}$$

the j-th hidden neuron embeds a bias term,

$$v_j^0, \quad j \in \{1, \ldots, h\}, \tag{3}$$

and a nonlinear activation function, $\varphi: \mathbb{R} \to \mathbb{R}$. Thus the neuron's response to an input stimulus, $\boldsymbol{x}$, is:

$$\varphi(\boldsymbol{v}_j \cdot \boldsymbol{x} + v_j^0), \quad j \in \{1, \ldots, h\}. \tag{4}$$

Note that Eq. (4) can be further generalized to include a wider class of functions [50], [51], [52]; therefore, the response of a neuron to an input stimulus $\boldsymbol{x}$ can be generically represented by any nonlinear piecewise continuous function characterized by a set of parameters. In ELM, these parameters ($\boldsymbol{v}_j$ and v_j^0) are set randomly. A vector of weighted links, $\boldsymbol{w} \in \mathbb{R}^h$, connects the hidden neurons to the output neuron without any bias. The overall output function, $f(\boldsymbol{x})$, of the network is:

$$f(\boldsymbol{x}) = \sum\nolimits_{j=1}^{h} \boldsymbol{w}_j \varphi(\boldsymbol{v}_j \cdot \boldsymbol{x} + v_j^0). \tag{5}$$

It is convenient to define an activation matrix, $V \in \mathbb{R}^{n \times h}$, such that the entry $V_{i,j}$ is the activation value of the j-th hidden neuron for the i-th input pattern. The V matrix is:

$$V = \begin{bmatrix} \varphi(\boldsymbol{v}_1 \cdot \boldsymbol{x}_1 + v_1^0) & \cdots & \varphi(\boldsymbol{v}_h \cdot \boldsymbol{x}_1 + v_h^0) \\ \vdots & \ddots & \vdots \\ \varphi(\boldsymbol{v}_1 \cdot \boldsymbol{x}_n + v_1^0) & \cdots & \varphi(\boldsymbol{v}_h \cdot \boldsymbol{x}_n + v_h^0) \end{bmatrix} = \begin{bmatrix} \phi^T(\boldsymbol{x}_1) \\ \vdots \\ \phi^T(\boldsymbol{x}_n) \end{bmatrix}. \tag{6}$$

In the ELM model, the quantities $\{\boldsymbol{v}_j, v_j^0\}$ in Eq. (4) are set randomly and are not subject to any adjustment, and the quantity $\boldsymbol{w}$ in Eq. (5) is the only degree of freedom. Hence, the training problem reduces to minimization of the convex cost:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \| V\boldsymbol{w} - \boldsymbol{y} \|^2. \tag{7}$$

A matrix pseudo-inversion yields the unique L_2 solution, as proven in [51], [53]:

$$\boldsymbol{w}^* = V^+ \boldsymbol{y}. \tag{8}$$

The simple, efficient procedure to train the ELM therefore involves the following steps:

1) Randomly generate hidden node parameters (in or case $\boldsymbol{v}_i$ and bias v_i^0) for each hidden neuron;
2) Compute the activation matrix V, of Eq. (6);
3) Compute the output weights by solving the pseudo-inverse problem of Eq. (8).

Despite the apparent simplicity of the ELM approach, the crucial result is that even random weights in the hidden layer endow a network with notable representation ability. Moreover, the theory derived in [53] proves that regularization strategies can further improve the approach's generalization performance. As a result, the cost function of Eq. (7) is augmented by a regularization factor as follows:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \| V\boldsymbol{w} - \boldsymbol{y} \|^2 \quad \text{and} \quad \| \boldsymbol{w} \|, \tag{9}$$

where $\| \boldsymbol{w} \|$ can be any suitable norm of the output weights [53]. A common approach is then to use the L_2 regularizer

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \| V\boldsymbol{w} - \boldsymbol{y} \|^2 + \lambda \| \boldsymbol{w} \|^2, \tag{10}$$

and consequently the vector of weights $\boldsymbol{w}^*$ is then obtained as follows:

$$\boldsymbol{w}^* = (V^T V + \lambda I)^{-1} V^T \boldsymbol{y}, \tag{11}$$

where $I \in \mathbb{R}^{h \times h}$ is an identity matrix.

V. ELM for Big Data on Spark

Spark [17] is a state-of-the-art framework for high performance memory parallel computing designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data [14], [17], [54]. One recent solution for Big Data analytics is the use of cloud computing [19], [55], [56], which makes available hundreds or thousands of machines to provide services such as computing and storage.

Various cluster management options are available for running Spark [57]. In this work, we chose to deploy Spark in a Hadoop cluster. The selected Hadoop architecture was composed of N_M slave machines and two additional machines that run as masters: one for controlling HDFS and the other for resource management.

The software packages installed on each machine were Hadoop 2.7.1 and Spark 1.5.1. In order to exploit this architecture, we had to modify the ELM formulation to cope with the main computational issues of ELM:

(I) computing the matrix V of Eq. (6);

(II) finding the solution of $\boldsymbol{w}^*$ of Eq. (11).

The main idea behind the Spark technology is that we have to reduce access to the disk as much as possible and make as much computation as possible in memory. Moreover, since Spark is designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data, it may not be efficient to compute the solution in the form of Eq. (11).

Hence, let us start from issue (II). Instead of solving the problem of Eq. (9) with the approach of Eq. (11), let us adopt a Stochastic Gradient Descent (SGD) algorithm. The SGD algorithm is a very general optimization algorithm, which is able to solve a problem efficiently in the following form:

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{n} C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i) + \lambda R(\boldsymbol{w}), \tag{12}$$

where $R(\boldsymbol{w})$ is a regularizer [58]–[61] and $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$ is a convex relaxation of the Hard Loss Function [45]. λ balances the tradeoff between the over- and underfitting tendency of the algorithm. Based on the choice of $R(\boldsymbol{w})$ and $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$, we can retrieve different algorithms. If $R(\boldsymbol{w}) = 0$ and $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i) = (\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i)^2$, we get the ELM formulation of Eq. (7) while if $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$ and $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i) = (\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i)^2$, we get the ELM formulation of Eq. (9). Other possible choices are: SVM [62], Regularized Least Squares [63], Least Squares SVM [64], Logistic Regression [65], Lasso [60], Elastic Net [61], etc. In Table 1 we report a series of possible choices of $R(\boldsymbol{w})$ and $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$. The SGD algorithm for optimizing Problem (12) is reported in Algorithm 1 [44]. In Algorithm 1, τ and n_{iter} are parameters related with the speed of the optimization algorithms. Therefore, usually τ and n_{iter} are set based on the experience of the user. In any case τ and n_{iter} can be seen as other regularization terms as λ since they are connected with the early stopping regularization technique [66], [67].

Algorithm 1 is well-suited for implementation in Spark and many of these tools are already available in MLib [68]. Basically, the implementation of Algorithm 1 reported in Algorithm 2 is an application of three functions: a filter (for the gradients that require an IF condition in Table 1), a map for the computation of the gradient and a reduction function for the sum of each single gradient.

The main problem of Algorithm 2 is the computation and storage of V. If $h \ll d$ it means that $V \in \mathbb{R}^{n \times h}$ will be much smaller than the dataset which belongs to $\mathbb{R}^{n \times d}$. In this case, it is more appropriate to compute it before the SGD algorithms

TABLE 1 Possible Regularizers and Convex Approximations of the Hard Loss Function.

(A) REGULARIZERS

	$R(\boldsymbol{w})$	$\frac{\partial}{\partial w_j} R(\boldsymbol{w})$
R1	0	0
R2	$\|\boldsymbol{w}\|_1$	$\text{sign}(w_j)$
R3	$\|\boldsymbol{w}\|_2$	$\frac{w_j}{\|\boldsymbol{w}\|_2}$
R4	$\eta\|\boldsymbol{w}\|_2 + (1-\eta)\|\boldsymbol{w}\|_1$, $\eta \in (0,1)$	$\eta\frac{w_j}{\|\boldsymbol{w}\|_2} + (1-\eta)\,\text{sign}(w_j)$
R5	$\|\boldsymbol{w}\|_p$, $p \in [0,\infty)$, $p > 1 \to$ convex	$\frac{w_j \lvert w_j\rvert^{p-2}}{\|\boldsymbol{w}\|_p^{p-1}}$

(B) CONVEX APPROXIMATIONS OF THE HARD LOSS FUNCTION

	$C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$	$\frac{\partial}{\partial w_j} C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$
L1	$\lvert\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i\rvert$	$\text{sign}(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i) x_{i,j}$
L2	$(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i)^2$	$2(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i) x_{i,j}$
L3	$\max[0, 1 - y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)]$	$\begin{cases} -y_i x_{i,j} & \text{if } y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \leq 1 \\ 0 & \text{otherwise} \end{cases}$
L4	$\{\max[0, 1 - y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)]\}^2$	$\begin{cases} -2y_i[1 - y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)] x_{i,j} \\ \quad \text{if } y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \leq 1 \\ 0 \quad \text{otherwise} \end{cases}$
L5	$\frac{\ln\{1 + \exp[-y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)]\}}{\ln(2)}$	$-y_i \left[1 - \frac{1}{1 + \exp[-y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)]}\right] x_{i,j}$

Algorithm 1 SGD for Eq. (12).

```
   Input: $\mathcal{D}_n$, $\lambda$, $\tau$, $n_{\text{iter}}$
   Output: $\boldsymbol{w}$
1  Read $\mathcal{D}_n$;
2  Compute $V$ (Eq. (6));
3  $w = 0$;
4  for $t \leftarrow 1$ to $n_{\text{iter}}$ do
5      $w = w - \frac{\tau}{\sqrt{t}} \frac{\partial}{\partial w}\left[\sum_{i=1}^{n} C(w, \phi(x_i), y_i) + \lambda R(w)\right]$;
6  return $(w, b)$;
```

Algorithm 2 SGD for Eq. (12) on Spark.

```
   Input: $\mathcal{D}_n$, $\lambda$, $\tau$, $n_{\text{iter}}$
   Output: $\boldsymbol{w}$
1  Read $\mathcal{D}_n$;
2  Compute $V$ (Eq. (6))
   /* Compute all the projection
      defined by $\phi$                          */
3  $w = 0$;
4  for $t \leftarrow 1$ to $n_{\text{iter}}$ do
5      $g = (V, y)$.filter(GradientCondition($w$))
       /* Apply the IF statement if
          required (Table 1)                     */
6      .map(Gradient())
       /* Compute the gradient                   */
7      .reduce(Sum())
       /* Sum all the gradients                  */
8      $w = w - \frac{\tau}{\sqrt{t}} g$;
9  return $w$;
```

start the iterative process and keep it in memory (note that the computation of V is fully parallel). In this way all the data $\mathbb{R}^{n\times d}$ projected by ϕ into matrix $V \in \mathbb{R}^{n\times h}$ can be largely kept in volatile memory (RAM) instead of reading from the disk. If instead $h \gg d$, we risk that $V \in \mathbb{R}^{n\times h}$ does not fit into the RAM and the consequent risk is to make too many accesses to the disk employing Algorithm 2. For this reason, we adopt two different strategies:

❏ if h is approximately the same magnitude or smaller than d, we use Algorithm 2 and we compute the matrix V at the beginning;

❏ if $h \gg d$, we adopt Algorithm 3 where $\phi(\boldsymbol{x}_i)$ is computed online in order to avoid to read the data from the disk.

Quite obviously, the limit is given by the size of the RAM of each node and the number of nodes. Until the algorithm is able to keep most of the data in memory, it is better to use Algorithm 2. Algorithm 3 allows us to partially reduce the effect of having to access the data on the disk by paying the price of computing $\phi(\boldsymbol{x}_i)$ online. In fact, Algorithm 3 does not precompute $V \in \mathbb{R}^{n\times h}$ at the beginning but it keeps in memory the data $\mathcal{D}_n$ and, at every iteration of the SGD algorithm, it computes online both the projection induced by ϕ and the gradient. Consequently, there is no need to store $V \in \mathbb{R}^{n\times h}$.

Algorithm 3 **SGD for Eq. (12) on Spark.**

```
  Input: $\mathcal{D}_n$, $\lambda$, $\tau$, $n_{\text{iter}}$
  Output: $\boldsymbol{w}$
1 Read $\mathcal{D}_n$;
2 $w = 0$;
3 for $t \leftarrow 1$ to $n_{\text{iter}}$ do
4         $g = \mathcal{D}_n$.filter(GradientCondition(w))
          /* the IF statement if required
             (see Table 1)                        */
5       .map($\phi$ & Gradient())
          /* Compute $\phi$ and the gradient       */
6       .reduce(Sum())
          /* Sum all the gradients                */
7       $w = w - \frac{\tau}{\sqrt{t}} g$;
8 return $\boldsymbol{w}$;
```

VI. Performance Assessment and Uncertainty Quantification

The selection of the optimal hyperparameters of a predictive model is the fundamental problem of STL, which is still the target of current research [28], [30], [69]–[72]. The approaches can be divided in two large families: Resampling methods; like Hold Out, Cross Validation, and the Non-parametric Bootstrap [30], [71], [73], [74], and more recent In-Sample methods; like the class of function-based methods [30] (based on the VC-Dimension [62], Rademacher Complexity (RC) [23], [24], [75], [76], PAC-Bayes Theory [77], [78]), and algorithm-based methods [26] (based on Compression Bounds [79], and Algorithmic Stability (AS) Theory [25], [80]).

Resampling methods [30], [81] are favored by practitioners because they work well in many situations and allow the application of simple statistical techniques for estimating the quantities of interest. Some examples of resampling methods are the well-known k-Fold Cross Validation (KCV), the Leave-One-Out (LOO), and the Non-parametric Bootstrap (BTS) [71], [74], [82].

In-Sample methods [30], [81], instead, allow the exploitation of a whole set of available data for both training the model and estimating its generalization error, thanks to the application of rigorous statistical procedures [25], [28], [78].

For more details about the advantages and disadvantages of the different methods one can refer to [26], [29], [30].

A. Resampling Methods

These techniques rely on a similar idea: the original dataset $\mathcal{D}_n$ is resampled once or many (n_r) times, with or without replacement, to build three independent datasets called training, validation, and test sets respectively, $\mathcal{L}_l^r$, $\mathcal{V}_v^r$, and $\mathcal{T}_t^r$, with $r \in \{1, \ldots, n_r\}$. Note that $\mathcal{L}_l^r \cap \mathcal{V}_v^r = \oslash$, $\mathcal{L}_l^r \cap \mathcal{T}_t^r = \oslash$, and $\mathcal{V}_v^r \cap \mathcal{T}_t^r = \oslash$. Then, in order to select the best set of hyperparameters $\mathcal{H}$ in a set of possible ones $\mathfrak{K} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$ for the algorithm $\mathcal{A}_{\mathcal{H}}$ or, in other words, to perform the Model Selection (MS), we have to apply the following procedure:

$$\mathcal{H}^* : \min_{\mathcal{H}\in\mathfrak{K}} \frac{1}{n_r}\sum_{r=1}^{n_r} \hat{L}_{\text{emp}}(\mathcal{A}_{(\mathcal{L}_l^r,\mathcal{H})}, \mathcal{V}_v^r). \tag{13}$$

Since the data in $\mathcal{L}_l^r$ are i.i.d. from the one in $\mathcal{V}_v^r$ the idea is that $\mathcal{H}^*$ should be the set of hyperparameters which allows one to achieve a small error on a dataset that is independent from the training set.

The uncertainty quantification, instead, is performed as follows:

$$L(\mathcal{A}_{(\mathcal{D}_n,\mathcal{H}^*)}) \le \frac{1}{n_r}\sum_{r=1}^{n_r} \hat{L}_{\text{emp}}(\mathcal{A}_{(\mathcal{L}_l^r\cup\mathcal{V}_v^r,\mathcal{H}^*)}, \mathcal{T}_t^r) + \sqrt{\frac{x}{2t}}, \tag{14}$$

where the bound holds with probability $(1 - e^{-x})$. Note that after the best set of hyperparameters is found, one can select the best model by training the algorithm with the whole dataset $\mathcal{A}_{(\mathcal{D}_n,\mathcal{H}^*)}$ [30], [69], [83]. Moreover, since the data in $\mathcal{L}_l^r \cup \mathcal{V}_v^r$ are i.i.d. with respect to $\mathcal{T}_t^r$ we have that $\hat{L}_{\text{emp}}(\mathcal{A}_{(\mathcal{L}_l^r\cup\mathcal{V}_v^r,\mathcal{H}^*)}, \mathcal{T}_t^r)$ is an unbiased estimator of $L(\mathcal{A}_{(\mathcal{D}_n,\mathcal{H}^*)})$. Then, we can use any concentration result, like the Hoeffding inequality [84], for bounding the bias between the expected value and its empirical estimator.

Note, also, that we get the hold-out method [30] if $n_r = 1$, if l, v, and t are set a priori such that $n = l + v + t$ and if the resample procedure is performed without replacement. For implementing the complete KCV, instead, we have to set $n_r \le \binom{n}{k}\binom{n-\frac{n}{k}}{k}$, $l = (k-2)(n/k)$, $v = n/k$, and $t = n/k$ and the resampling must be done without replacement [30], [69], [71]. Finally, for implementing the Non-parametric Bootstrap, $l = n$ and $\mathcal{L}_l^r$ must be sampled with replacement from $\mathcal{D}_n$, while $\mathcal{V}_v^r$ and $\mathcal{T}_t^r$ are sampled without replacement from the sample of $\mathcal{D}_n$ that has not been sampled in $\mathcal{L}_l^r$ [30], [74]. Note that for the Non-parametric Bootstrap procedure $n_r \leq \binom{2n-1}{n}$.

It is worthwhile noting that the only hypothesis needed in order to rigorously apply the resampling technique is the i.i.d. hypothesis on the data in $\mathcal{D}_n$ and that all these techniques work for any deterministic algorithm.

B. In-Sample Methods

For the In-Sample methods, two subfamilies of techniques are identified: the class of function-based ones and algorithm-based ones [26]. The difference between the two classes is that the function-based techniques require the knowledge of $\mathcal{F}_\mathcal{H}$ and thus, cannot be applied to some algorithms (e.g., the kNN algorithm), while the algorithm-based techniques can be applied to any deterministic algorithm without any additional knowledge. Both subfamilies, like the resampling methods, require the i.i.d. hypothesis.

One of the most powerful techniques in the class of function-based techniques is based on the Rademacher Complexity [23], [30]. In particular, for any bounded loss function $\ell_b(f,z) \in [0,1]$ it is possible to prove that the following bound holds with probability $(1 - 2e^{-x})$ [85]:

$$L(f) \leq \hat{L}_{\text{emp}}(f, \mathcal{D}_n) + \hat{R}_n(\mathcal{F}_\mathcal{H}) + 3\sqrt{\frac{x}{2n}}, \quad \forall f \in \mathcal{F}_\mathcal{H} \tag{15}$$

where

$$\hat{R}_n(\mathcal{F}_\mathcal{H}) = \mathbb{E}_\sigma \sup_{f \in \mathcal{F}_\mathcal{H}} \frac{2}{n} \sum_{i=1}^{n} \sigma_i \ell_b(f, z_i), \tag{16}$$

$$\sigma_{i \in \{1,\ldots,n\}} \in \{\pm 1\}, \quad \mathbb{P}\{\sigma_i = +1\} = \mathbb{P}\{\sigma_i = -1\} = \frac{1}{2}.$$

Therefore, based on the Structural Risk Minimization principle [22], one can design a series of function classes of increasing size, $\mathfrak{F} = \{\mathcal{F}_{\mathcal{H}_1}, \mathcal{F}_{\mathcal{H}_2}, \ldots\}$ with $\mathcal{F}_{\mathcal{H}_1} \subseteq \mathcal{F}_{\mathcal{H}_2} \subseteq \cdots$, so to compute at the same time both the MS and the uncertainty quantification:

$$f^*, \mathcal{F}_{\mathcal{H}^*}: L(f^*) \leq \min_{\mathcal{F}_\mathcal{H} \in \mathfrak{F}} \left[\hat{L}_{\text{emp}}(f, \mathcal{D}_n) + \hat{R}_n(\mathcal{F}_\mathcal{H}) + \sqrt{\frac{9x}{2n}} \right]. \tag{17}$$

Unfortunately, the quantity of Eq. (16) cannot be computed if we do not know $\mathcal{F}_\mathcal{H}$. Moreover, for many algorithms it is impossible to define $\mathcal{F}_\mathcal{H}$ [26]. Algorithm-based techniques circumvent this problem through the concept of Algorithmic Stability [25], [26], [86]. In particular, for any bounded loss function ℓ_b it is possible to prove that the following bounds hold with probability $(1 - \delta)$ [25]:

$$L(f) \leq \hat{L}_{\text{emp}}(f, \mathcal{D}_n) + \sqrt{\frac{1}{2n\delta} + \frac{3\beta_{\text{emp}}}{\delta}}, \tag{18}$$

$$L(f) \leq \hat{L}_{\text{loo}}(f, \mathcal{D}_n) + \sqrt{\frac{1}{2n\delta} + \frac{3\beta_{\text{loo}}}{\delta}}, \tag{19}$$

where f: $\mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}$ and

$$\beta_{\text{emp}}(\mathcal{A}_\mathcal{H}, n) = \mathbb{E}_{\mathcal{D}_n, z_i'} \left| \ell_b(A_{(\mathcal{D}_n, \mathcal{H})}, z_i) - \ell_b(A_{(\mathcal{D}_n^i, \mathcal{H})}, z_i) \right|,$$

$$\beta_{\text{loo}}(\mathcal{A}_\mathcal{H}, n) = \mathbb{E}_{\mathcal{D}_n, z} \left| \ell_b(A_{(\mathcal{D}_n, \mathcal{H})}, z) - \ell_b(A_{(\mathcal{D}_n^{\setminus i}, \mathcal{H})}, z) \right|.$$

The bounds of Eqns. (18) and (19) are polynomial bounds in n (not very tight indeed when n is small) while β_{emp} and β_{loo} are two versions of Hypothesis Stability which are able to take into account both the properties of the algorithm and the property of the distribution that has generated the data $\mathcal{D}_n$ [25], [26]. It is possible to improve the bounds of Eqns. (18) and (19) by exploiting a stronger notion of algorithmic stability, known as the Uniform Stability. In particular, the following bounds hold with probability $(1 - \delta)$:

$$L(f) \leq \hat{L}_{\text{emp}}(f, \mathcal{D}_n) + 2\beta^i + (4n\beta^i + 1)\sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{2n}}, \tag{20}$$

$$L(f) \leq \hat{L}_{\text{loo}}(f, \mathcal{D}_n) + \beta^{\setminus i} + (4n\beta^{\setminus i} + 1)\sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{2n}}, \tag{21}$$

where $f: \mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}$ and

$$\beta^i(\mathcal{A}_\mathcal{H}, n) = \left| \ell(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}, \cdot) - \ell(\mathcal{A}_{(\mathcal{D}_n^i, \mathcal{H})}, \cdot) \right|_\infty, \tag{22}$$

$$\beta^{\setminus i}(\mathcal{A}_\mathcal{H}, n) = \left| \ell(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}, \cdot) - \ell(\mathcal{A}_{(\mathcal{D}_n^{\setminus i}, \mathcal{H})}, \cdot) \right|_\infty. \tag{23}$$

Unfortunately, the Uniform Stability (β^i or $\beta^{\setminus i}$) is not able to take into account the properties of the distribution that has generated the data $\mathcal{D}_n$ and is sometimes unable to capture the properties of the algorithm because it deals with a worst-case learning scenario [26]. Nevertheless, all the four stability-based bounds of Eqns. (18), (19), (20), and (21) can be used to select the best set of hyperparameters $\mathcal{H}$ in a set of possible one $\mathfrak{H} = \{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$ for the algorithm $\mathcal{A}_\mathcal{H}$. In particular, all the bounds are expressed in the form: $L(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}) \leq \epsilon(\mathcal{A}_\mathcal{H}, \mathcal{D}_n, n, \delta p_\mathcal{H})$. Thus, in order to perform the MS procedure and uncertainty quantification, we have to aprioristically assign to each set of hyperparameters a probability $p_\mathcal{H}$ (where $\sum_{i=1}^{\cdots} p_{\mathcal{H}_i} = 1$) of being chosen during the MS procedure. The algorithmic stability-based MS and uncertainty quantification procedure can then be summarized as follows:

$$\mathcal{A}_{(\mathcal{D}_n, \mathcal{H}^*)}, \mathcal{H}^*: L(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H}^*)}) \leq \min_{\mathcal{H} \in \mathfrak{H}} \epsilon(\mathcal{A}_\mathcal{H}, \mathcal{D}_n, n, \delta p_{\mathcal{H}^*}). \tag{24}$$

The procedure of Eq. (24) can be exploited with any algorithm for which it is possible to compute one of the notions of stability.

VII. Computational Issues for Big Data Analytics

Both naive resampling and In-Sample methods are computationally expensive when the number of samples is large [30], [72]. For this reason, we will focus on adapting these techniques to the Big Data context.

A. Bag of Little Bootstraps

The standard Non-parametric Bootstrap procedure requires, $\forall \mathcal{H} \in \{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$, to train many (n_r) models, and is computationally very expensive if n is large. For this reason the Bag of

Little Bootstraps approach [87]–[90] represents an alternative to standard Non-parametric Bootstrap; it considers only $b = n^\gamma$ data, with $\gamma \in [1/2, 1]$, in place of the whole dataset during the creation of the train, validation and test sets. Note that $\gamma \in [1/2, 1]$ is necessary to maintain the statistical property of the procedure. In particular, the Bag of Little Bootstraps [87] consists in sampling $n_r^{\text{no-rep}}$ times from $\mathcal{D}_n$ without replacement, several datasets $\mathcal{B}_b^i$ with $i \in \{1, \dots, n_r^{\text{no-rep}}\}$, consisting of $b \in [\sqrt{n}, n]$ samples. Then, each $\mathcal{B}_b^i$ is sampled with replacement $n_r^{\text{yes-rep}}$ times, in order to derive $\mathcal{L}_n^{i,j}$ datasets with $j \in \{1, \dots, n_r^{\text{yes-rep}}\}$, each consisting of n samples. All the samples of $\mathcal{D}_n$, or parts of them, that have not been sampled in $\mathcal{L}_n^{i,j}$ are used as validation set and test set $\mathcal{V}_v^{i,j} \subseteq \mathcal{D}_n \backslash \mathcal{L}_n^{i,j}, \mathcal{T}_t^{i,j} \subseteq \mathcal{D}_n \backslash \mathcal{L}_n^{i,j}$ and $\mathcal{V}_v^{i,j} \cap \mathcal{T}_t^{i,j} = \emptyset$. Finally, the models are trained on the sets $\mathcal{L}_n^{i,j}$ and tested on the corresponding $\mathcal{V}_v^{i,j}$; thus we define the following MS procedure:

$$\mathcal{H}^*: \min_{\mathcal{H} \in \mathfrak{H}} \frac{1}{n_r^{\text{no-rep}}} \frac{1}{n_r^{\text{yes-rep}}} \sum_{i=1}^{n_r^{\text{no-rep}}} \sum_{j=1}^{n_r^{\text{yes-rep}}} \hat{L}_{\text{emp}}(\mathcal{A}_{(\mathcal{L}_n^{i,j}, \mathcal{H})}, \mathcal{V}_v^{i,j}). \quad (25)$$

Note that in order to find $\mathcal{H}^*$ with the procedure of Eq. (25), we have to train a series of models over sets composed by a maximum of n^γ distinct samples. This means that the MS strategy, if n is large with respect to $n_r^{\text{no-rep}} n_r^{\text{yes-rep}}$, scales with n^γ. Therefore, the procedure scales sub-linearly with respect to n, and in the best case scenario, scales with $O(\sqrt{n})$. Analogous to the usual Non-parametric Bootstrap procedure, the uncertainty quantification is performed as follows:

$$L(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H}^*)}) \leq \frac{1}{n_r} \sum_{r=1}^{n_r} \hat{L}_{\text{emp}}(\mathcal{A}_{(\mathcal{L}_n^{i,j}, \mathcal{H}^*)}, \mathcal{T}_t^{i,j}) + \sqrt{\frac{x}{2t}}, \quad (26)$$

where the best model is obtained by training the algorithm with the whole dataset [83]: $f^* = \mathcal{A}_{(\mathcal{D}_n, \mathcal{H}^*)}$.

Finally, we would like to underline that γ balances the tradeoff between accuracy and computational requirements of the statistical procedure [88], [90]. The more $\gamma \to 1$, the better the MS strategy will perform. Since we deal with Big Data in this paper, we set $\gamma = 1/2$. The application of this approach to ELM is straightforward by noting that the hyperparameters of ELM are $\lambda \in [0, \infty)$ and $h \in \{1, 2, \dots\}$.

B. Simplified Rademacher Complexity

Now, we show that the Rademacher Complexity of ELM (which employs the general regularization schema of Eq. (12)) can be easily upper bounded. In particular, let us truncate the loss functions such that $\ell_T(f, \boldsymbol{z}) = \min[1, C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)]$. It is easy to see that $\ell_H(f, \boldsymbol{z}) \leq \ell_T(f, \boldsymbol{z})$. Consequently, the generalization error computed with $\ell_H(f, \boldsymbol{z})$ is equal to or less than the one computed with $\ell_T(f, \boldsymbol{z})$. By exploiting the bound of Eq. (15) for $\ell_T(f, \boldsymbol{z})$ the computation of the empirical error is straightforward and it is possible to prove that the Rademacher Complexity can be upper bounded as follows [85]:

$$\begin{aligned} \hat{R}_n(\mathcal{F}_\mathcal{H}) &= \mathbb{E}_\sigma \sup_{f \in \mathcal{F}_\mathcal{H}} \frac{2}{n} \sum_{i=1}^n \sigma_i \ell_T(f, \boldsymbol{z}_i) \\ &\leq \frac{L}{n} \sqrt{\sum_{i=1}^n \| \boldsymbol{w}_{\lambda,h}^* \|^2 \| \phi(\boldsymbol{x}_i) \|^2}, \end{aligned} \quad (27)$$

where L is the Lipschitz constant characterizing $\ell_T(f, \boldsymbol{z})$ and $(\boldsymbol{w}_{\lambda,h}^*)$ is the solution to the ELM problem of Eq. (10) (or more generally Eq. (12)) for a given h and λ. $C(\boldsymbol{w}, \phi(\boldsymbol{x}_i), y_i)$ and $R(\boldsymbol{w})$ can be any of the ones reported in Table 1. Note that h and λ define the size of the class of functions in ELM [30] and thus, we can plug this result in the procedure of Eq. (17) and obtain a computationally efficient way of assessing the performance and quantifying the uncertainty of ELM. In fact, in order to exploit the procedure of Eq. (17), it is only necessary to train, for each values of h and λ, the ELM model and to compute the quantity of Eq. (27) which is computationally inexpensive once the ELM has been trained.

C. Simplified Uniform Stability

In this section, we show how to apply the bound based on the Uniform Stability in the Big Data scenario. The bound of Eq. (21), which takes into account the leave-one-out error, is too computationally expensive to compute. Instead, we employ that of Eq. (20). As in Section (VII-B), we use the truncated loss function since for the hard loss function we have trivially that $\beta^i(\mathcal{A}_\mathcal{H}, n) = 1$. Consequently, once the ELM has been trained we can compute the empirical error with the truncated loss. Computing $\beta^i(\mathcal{A}_\mathcal{H}, n)$ is not easy but, thanks to the result of [25], it is possible to upper bound it in the case of ELM as follows:

$$\begin{aligned} \beta^i(\mathcal{A}_\mathcal{H}, n) &= | \ell(\mathcal{A}_{(\mathcal{D}_n, \mathcal{H})}, \cdot) - \ell(\mathcal{A}_{(\mathcal{D}_n^i, \mathcal{H})}, \cdot) |_\infty \\ &\leq \frac{L \max\{\| \phi(\boldsymbol{x}_1) \|^2, \dots, \| \phi(\boldsymbol{x}_n) \|^2\}}{n\lambda}, \end{aligned} \quad (28)$$

where L is the Lipschitz constant characterizing $\ell_T(f, \boldsymbol{z})$. In this case, $R(\boldsymbol{w})$ must be $\| \boldsymbol{w} \|_2$ since the bound does not hold for all the $R(\boldsymbol{w})$ reported in Table 1. Then the application of the procedure of Eq. (24) to ELM becomes straightforward and computationally inexpensive. From Eq. (28) it seems that the Uniform Stability takes into account only the property of ELM through λ and h through ϕ, and not the distribution of the data. In other words, the Uniform Stability upper bound of Eq. (28) is not able to capture the effect of changing the loss.

D. Bag of Little Hypothesis Stabilities

In order to overcome the issues of the Uniform Stability, we exploit the proposal of [26] to estimate the Hypothesis Stability instead. As we will see, this proposal is also well suited for Big Data applications. As for the Uniform Stability, we do not consider the bound of Eq. (19) since it is too computationally expensive. Consequently, we take into account the bound of Eq. (18). In this case, we do not need to exploit the truncated loss function, but can use the hard loss function directly once the ELM model has been trained with a fixed value of λ. In order to compute the bound of Eq. (18) and perform the procedure of Eq. (28), we need to compute $\beta_{\text{emp}}(\mathcal{A}_\mathcal{H}, n)$. We start by making an assumption on the learning algorithm $\mathcal{A}_\mathcal{H}$. In particular, we suppose that the Hypothesis Stability does not increase with the cardinality of the training set:

$$\beta_{\text{emp}}(\mathcal{A}_\mathcal{H}, n) \leq \beta_{\text{emp}}(\mathcal{A}_\mathcal{H}, \sqrt{n} - 1). \quad (29)$$

Affective analogical reasoning can be defined as the intrinsically human capacity to interpret the cognitive and affective information associated with natural language.

We point out that this property is a desirable requirement for any learning algorithm, because in order to be able to prove the learnability in the stability framework, we need that:

$$\lim_{n\to\infty}\beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, n) = 0, \tag{30}$$

or, in other words, that the impact on the learning procedure of removing or replacing one sample from $\mathcal{D}_n$ should decrease, on average, as n grows. Numerous researchers have already studied this property in the past. In particular, it is related to the consistency concept [46]. However, connections can also be identified with the trend of the learning curves of an algorithm [91]. Moreover, such quantity is also strictly linked to the concept of Smart Rule [46]. It is worth underlining that, in many of the above-referenced works, the property of Eq. (29) is proved to be satisfied by many well known algorithms (SVM, Regularized Least Squares and consequently ELM, k-Local Rule with $k > 1$, etc.).

Let us define $\hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, n, \mathcal{D}_n)$ which is an unbiased estimator of $\beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, n)$:

$$\hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1, \mathcal{D}_n) = \frac{1}{\sqrt{n}}\sum_{k=1}^{\sqrt{n}}\left|\ell(\mathcal{A}_{\breve{\mathcal{D}}^k_{\sqrt{n}-1}}, \breve{z}^k) - \ell(\mathcal{A}_{(\breve{\mathcal{D}}^k_{\sqrt{n}-1})^{\setminus i}}, \breve{z}^k)\right|, \tag{31}$$

where $i \in \{1, \ldots, \sqrt{n}-1\}$. Moreover:

$$\breve{\mathcal{D}}^k_{\sqrt{n}-1} : \{z_{(k-1)\sqrt{n}+1}, \ldots, z_{(k-1)\sqrt{n}+\sqrt{n}-1}\}, \tag{32}$$

$$\breve{z}^k : z_{(k-1)\sqrt{n}}. \tag{33}$$

By construction we have that $\hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1, \mathcal{D}_n)$ is an unbiased estimator of $\beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1)$:

$$\mathbb{E}_{\mathcal{D}_n}\hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1, \mathcal{D}_n) = \beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1). \tag{34}$$

Since all the quantities in the summations of Eq. (31) are $\{\pm 1\}$ valued i.i.d. random variables (since they are computed over different sets of data) extracted from a Bernoulli distribution of mean $\beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1)$, we have that the following bound holds [84] with probability $(1-e^{-x})$:

$$\beta_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1) \le \hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1, \mathcal{D}_n) + \sqrt{\frac{x}{2\sqrt{n}}}. \tag{35}$$

Note that plugging Eq. (35) into the bound of Eq. (18) gives a fully empirical bound where all the quantities can be computed from the data [26]. In particular, once the ELM has been trained for a given h and λ, the empirical error, computed with the hard loss function, is trivially computable, while $\hat{\beta}_{\text{emp}}(\mathcal{A}_{\mathcal{H}}, \sqrt{n}-1, \mathcal{D}_n)$ requires the training of many ELMs on a small subset of the data $(\sqrt{n})$, which is computationally inexpensive. Moreover, all these ELMs can be trained in parallel (see Eq. (31)). The application of the procedure of Eq. (24) to ELM then becomes straightforward. Note that, from Eq. (31), the hypothesis stability is able to capture both the property of the algorithm and the property of the distribution that has generated the data [26].

VIII. Affective Analogical Reasoning Dataset

The proposed approach has been tested on two affective analogical reasoning datasets. Affective analogical reasoning can be defined as the intrinsically human capacity to interpret the cognitive and affective information associated with natural language [92]. In particular, we employed two benchmarks, each one composed by 21743 common-sense concepts; each concept is represented according to the AffectiveSpace model [93] and the AffectiveSpace 2 model [94]. Both models are obtained as a vector space representation of the AffectNet network, a semantic network in which common-sense concepts (e.g., 'read book', 'payment', 'play music') are linked to a hierarchy of affective domain labels (e.g., 'joy', 'amazement', 'fear', 'admiration'). In this way, concepts conveying similar semantic and affective information, e.g., 'enjoy conversation' and 'chat with friends', tend to fall near each other in the multi-dimensional space. Both AffectNet and AffectiveSpace are publicly available at http://sentic.net. The difference between the two models is the following:

❑ AffectiveSpace is obtained applying principal component analysis (PCA) on the matrix representation of AffectNet [93].
❑ AffectiveSpace 2 is obtained applying a refined projection on the matrix representation of AffectNet [94].

In both cases, common-sense concepts are eventually represented by vectors of M coordinates. This number indicates the dimensionality of the AffectiveSpace and represents the trade-off between efficiency and precision: the bigger is M, the more precisely AffectiveSpace represents AffectNet's knowledge, but generating the vector space is slower, while the smaller is M, the more efficiently AffectiveSpace can be obtained. As already mentioned, concepts with the same affective orientation are likely to have similar features; i.e., concepts conveying the same emotion tend to fall near each other in AffectiveSpace. Concept similarity does not depend on their absolute positions in the vector space, but rather on the angle they make with the origin [95].

The Hourglass of Emotions [95] is employed to reason on the disposition of concepts in AffectiveSpace. In the model, affective states are represented by four concomitant but independent dimensions (Pleasantness, Attention, Sensitivity and Aptitude), which determine the intensity of the expressed/perceived emotion. Therefore, a four-dimensional vector can potentially synthesize the level of activation of each affective dimension of a concept. Beyond emotion detection, the Hourglass model is also used for polarity detection tasks. Polarity is defined in terms of the four affective dimensions, according to the formula:

$$p = \sum_{i=1}^{N} \frac{P(c_i) + |At(c_i)| - |S(c_i)| + Ap(c_i)}{3N} \quad (36)$$

where P is the pleasantness, At the attention, S the sensitivity, Ap the aptitude, c_i an input concept, N the total number of concepts, and 3 the normalization factor (as the Hourglass dimensions are defined as floats $\in [-1, 1]$). In the equation, Attention is taken as absolute value since both its positive and negative intensity values correspond to positive polarity values (e.g., 'surprise' is negative in the sense of lack of Attention, but positive from a polarity point of view). Similarly, Sensitivity is taken as negative absolute value since both its positive and negative intensity values correspond to negative polarity values (e.g., 'anger' is positive in the sense of level of activation of Sensitivity, but negative in terms of polarity). The publicly available Sentic API (on http://sentic.net/api) was used to obtain for each concept the level of each affective dimension.

According to the Hourglass model, the Sentic API expresses the levels as numbers $\in [-1, 1]$, which are eventually mapped into the associated polarity according to Eq. (36). In order to perform a binary classification task for each affective dimension and polarity, the values are then discretized: +1 for positive values and –1 for negative ones.

The experiments eventually involve two tasks:

- Classification of each affective dimension level and polarity detection for concepts expressed according to AffectiveSpace 1 [93];
- Classification of each affective dimension level and polarity detection for concepts expressed according to AffectiveSpace 2 [94];

In both cases, the dimension of the space M has been set equal to 100.

TABLE 2 Error (in percentage) on the reference set exploiting different losses and different MS strategies on AffectiveSpace 1.

ELMs	MS METHOD			
LOSS	BLB	SRC	SUS	BLHS
		PLEASANTNESS		
L1	5.32 ± 0.16	5.95 ± 0.18	5.96 ± 0.19	**4.76 ± 0.14**
L2	5.85 ± 0.18	6.59 ± 0.21	6.57 ± 0.21	**5.30 ± 0.17**
L3	4.75 ± 0.14	5.21 ± 0.17	5.31 ± 0.16	**4.16 ± 0.13**
L4	5.28 ± 0.16	5.92 ± 0.18	5.92 ± 0.19	**4.75 ± 0.15**
L5	5.36 ± 0.17	5.88 ± 0.19	5.89 ± 0.19	**4.77 ± 0.14**

TABLE 3 Error (in percentage) on the reference set exploiting different losses and different MS strategies on AffectiveSpace 2.

ELMs	MS METHOD			
LOSS	BLB	SRC	SUS	BLHS
		PLEASANTNESS		
L1	3.53 ± 0.11	3.93 ± 0.12	3.89 ± 0.12	**3.14 ± 0.10**
L2	3.82 ± 0.12	4.33 ± 0.14	4.32 ± 0.13	**3.48 ± 0.10**
L3	3.11 ± 0.10	3.46 ± 0.11	3.47 ± 0.11	**2.74 ± 0.09**
L4	3.45 ± 0.11	3.90 ± 0.12	3.93 ± 0.13	**3.13 ± 0.10**
L5	3.54 ± 0.11	3.83 ± 0.12	3.92 ± 0.12	**3.14 ± 0.09**

TABLE 4 Training time (in minutes) when different losses and different MS strategies are exploited on AffectiveSpace 1.

ELMs	MS METHOD			
LOSS	BLB	SRC	SUS	BLHS
		PLEASANTNESS		
L1	15.08 ± 1.09	**10.01 ± 0.76**	10.04 ± 0.71	18.03 ± 1.27
L2	15.10 ± 1.10	**10.01 ± 0.77**	10.07 ± 0.73	18.10 ± 1.22
L3	15.04 ± 1.09	10.06 ± 0.77	**10.05 ± 0.70**	18.10 ± 1.31
L4	15.07 ± 1.08	**10.05 ± 0.73**	**10.05 ± 0.71**	18.08 ± 1.20
L5	15.03 ± 1.00	**10.01 ± 0.76**	10.05 ± 0.72	18.11 ± 1.20

IX. Experimental Results

In this section[1], we show the results of applying the ELMs models described in Section V to the Affective Analogical Reasoning datasets described in Section VIII, where the performance of the models has been assessed by using the MS strategies described in Section VII.

In Tables 2 and 3 we have reported, respectively for AffectiveSpace 1 and AffectiveSpace 2 and for the Pleasantness, the error on the reference set of the ELMs model selected by exploiting regularizer $\|\boldsymbol{w}\|_2$, different losses (L1, …, L5 in Table 1), and different MS strategies (Bag of Little Bootstraps-BLB, Simplified Rademacher Complexity-SRC, Simplified Uniform Stability-SUS, and Bag of Little Hypothesis Stabilities-BLHS). In Table 4, for AffectiveSpace 1, we have reported the time required to build the ELMs model selected by exploiting different losses and different MS strategies. In particular, we reported only the time required for the Pleasantness task.

From Tables 2, 3, and 4 we can state that:

- AffectiveSpace 2 is able to better predict the affective dimensions and polarity with respect to AffectiveSpace 1.
- BLHS is the best method to perform MS since it is the one that more often selects the most accurate model according

[1]We do not report all the details and experiments because of space constraints, all the details can be found in the technical report available at http://sentic.net/slt-based-elm-for-big-social-data-analysis.pdf.

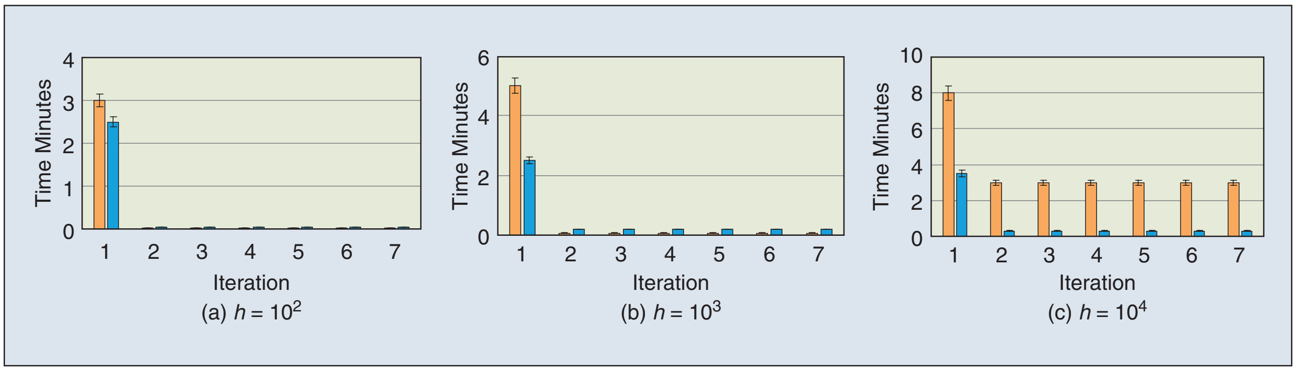

FIGURE 1 Comparison between algorithms 2 (orange) and 3 (blue): time needed to execute the first iteration (similarly to what is done in [17], [18]) and time of the next iterations. Results are averaged over 30 different realizations.

to the reference set. BLB performs well, while SRC and SUS offer the poorest performance.

❑ SRC and SUS are the most computationally saving methods, while the method that is more computational demanding is BLB (which in return, however, is also the most accurate one).

❑ The L3 loss function results to be the best loss for this task.

Note that all the methods perform quite well in practice and reach similar performance when n is large and, at the same time, are almost equally computationally expensive.

Finally, we compare the execution time between Algorithm 2 and 3. In particular, for ELMs with regularizer $\|\boldsymbol{w}\|_2$, loss L2 and $\lambda = 1$:

❑ Figure 1(a) reports for $h = 100$ and for Algorithms 2 and 3 on the time needed to execute the first iteration (similarly to what has been done in [17], [18]) and the time of the next iterations (results are averaged over 30 different realizations).

❑ Figure 1(b) reports on the same information for $h = 1{,}000$.

❑ Figure 1(c) reports on the same information for $h = 10{,}000$.

From Figures 1(a), 1(b) and 1(c) it is possible to state that:

❑ As expected, when h is smaller or comparable to d, we have that Algorithm 2 is the one with the best performance.

❑ When h becomes larger than d, the data stop to fit into memory; this increases the number of accesses to the disk for Algorithm 2 and consequently, the time needed to execute each iteration. Subsequently, Algorithm 3 becomes more efficient.

X. Conclusion

In this paper, we proposed an efficient implementation of the ELMs on Spark, in order to exploit the benefits of the Spark framework, in the context of big social data analysis. In particular, an approach to support emotion recognition and polarity detection in natural language text has been proposed and evaluated.

We also showed how to carefully assess the performance with the use of the most recent results from SLT. Unlike other statistical inference frameworks, SLT implements a worst-case approach to these problems, which allows for the obtaining of rigorous and consistent generalization bounds that can be exploited for assessing the performance of the ELMs. Thanks to recent advances, as presented in this paper, the computational requirements of these methods have been improved to allow for the scaling to large datasets, which are typical of Big Data applications.

Additional work in this direction is needed. In particular, other big data architectures are available with higher efficiency but lower fault tolerance (e.g., the one based on MPI and OpenMP [18]). It will also be interesting to extend these approaches to a semi-supervised setting since in Big Social Data Analysis more and more data are becoming available but just a small amount is supervised [96].

References

[1] S. Mills, S. Lucas, L. Irakliotis, M. Rappa, T. Carlson, and B. Perlowitz, (2012). Demystifying big data: A practical guide to transforming the business of government. [Online]. Technical report. Available: http://www. ibm. com/software/data/demystifying-big-data

[2] E. Cambria, H. Wang, and B. White, "Guest editorial: Big social data analysis," *Knowledge-Based Syst.*, vol. 69, pp. 1–2, 2014.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[4] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Clarendon Press, 1995.

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[6] L. Breiman, "Random forests," *Machine Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[7] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, 2014.

[8] Y. You, S. L. Song, H. Fu, A. Marquez, M. M. Dehnavi, K. Barker, K. W. Cameron, A. P. Randles, and G. Yang, "Mic-svm: Designing a highly efficient support vector machine for advanced modern multi-core and many-core architectures," in *Proc. IEEE Int. Parallel and Distributed Process. Symp.*, 2014.

[9] G. Huang, G. B. Huang, S. Song, and K. You, "Trends in extreme learning machines: A review," *Neural Networks*, vol. 61, pp. 32–48, 2015.

[10] J. Xin, Z. Wang, C. Chen, L. Ding, G. Wang, and Y. Zhao, "ELM*: Distributed extreme learning machine with mapreduce," *World Wide Web*, vol. 17, no. 5, pp. 1189–1204, 2014.

[11] Q. He, T. Shang, F. Zhuang, and Z. Shi, "Parallel extreme learning machine for regression based on mapreduce," *Neurocomputing*, vol. 102, pp. 52–58, 2013.

[12] S. Huang, B. Wang, J. Qiu, J. Yao, G. Wang, and G. Yu, "Parallel ensemble of online sequential extreme learning machine based on mapreduce," in *Proc. ELM-2014*, 2015.

[13] L. J. Cao, S. S. Keerthi, C. J. Ong, J. Q. Zhang, U. Periyathamby, X. J. Fu, and H. P. Lee, "Parallel sequential minimal optimization for the training of support vector machines," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 1039–1049, 2006.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX Conf. Networked Systems Design and Implementation*, 2012.

[15] K. Olukotun, "Beyond parallel programming with domain specific languages," in *Proc. Symp. Principles and Practice of Parallel Programming*, 2014.

[16] A. Akusok, K. M. Bjork, Y. Miche, and A. Lendasse, "High-performance extreme learning machines: A complete toolbox for big data applications," *IEEE Access*, vol. 3, pp. 1011–1025, 2015.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. USENIX Conf. Hot Topics in Cloud Computing*, 2010.

[18] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita, "Big data analytics in the cloud: Spark on hadoop vs MPI/openMP on Beowulf," *Procedia Comp. Sci.*, vol. 53, pp. 121–130, Aug. 2015.

[19] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *Proc. Int. Conf. Extending Database Technology*, 2011.
[20] L. Breiman, "Statistical modeling: The two cultures (with comments and a rejoinder by the author)," *Stat. Sci.*, vol. 16, no. 3, pp. 199–231, 2001.
[21] V. Dhar, "Data science and prediction," *Commun. ACM*, vol. 56, no. 12, pp. 64–73, 2013.
[22] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.
[23] V. Koltchinskii, "Rademacher penalties and structural risk minimization," *IEEE Trans. Inform. Theory*, vol. 47, no. 5, pp. 1902–1914, 2001.
[24] P. L. Bartlett, O. Bousquet, and S. Mendelson, "Local rademacher complexities," *Ann. Stat.*, vol. 33, no. 4, pp. 1497–1537, 2005.
[25] O. Bousquet and A. Elisseeff, "Stability and generalization," *J. Machine Learn. Res*, vol. 2, pp. 499–526, 2002.
[26] L. Oneto, A. Ghio, S. Ridella, and D. Anguita, "Fully empirical and data-dependent stability-based bounds," *IEEE Trans. Cybernetics*, vol. 45, no. 9, pp. 1913–1926, 2015.
[27] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[28] P. L. Bartlett, S. Boucheron, and G. Lugosi, "Model selection and error estimation," *Machine Learn.*, vol. 48, no. 1-3, pp. 85–113, 2002.
[29] J. Langford, "Tutorial on practical prediction theory for classification," *J. Machine Learn. Res.*, vol. 6, no. 1, pp. 273, 2006.
[30] D. Anguita, A. Ghio, L. Oneto, and S. Ridella, "In-sample and out-of-sample model selection and error estimation for support vector machines," *IEEE Trans. Neural Network Learn. Syst.*, vol. 23, no. 9, pp. 1390–1406, 2012.
[31] E. Cambria and B. White, "Jumping NLP curves: A review of natural language processing research," *IEEE Comput. Intell. Mag.*, vol. 9, no. 2, pp. 48–57, 2014.
[32] E. Cambria, B. Schuller, B. Liu, H. Wang, and C. Havasi, "Knowledge-based approaches to concept-level sentiment analysis," *IEEE Intell. Syst.*, vol. 28, no. 2, pp. 12–14, 2013.
[33] E. Cambria, B. Schuller, B. Liu, H. Wang, and C. Havasi, "Statistical approaches to concept-level sentiment analysis," *IEEE Intell. Syst.*, vol. 28, no. 3, pp. 6–9, 2013.
[34] A. Gangemi, V. Presutti, and D. Reforgiato, "Frame-based detection of opinion holders and topics: a model and a tool," *IEEE Comput. Intell. Mag.*, vol. 9, no. 1, pp. 20–30, 2014.
[35] C. Strapparava and A. Valitutti, "WordNet-Affect: An affective extension of WordNet," in *Proc. Int. Conf. Language Resources and Evaluation*, 2004.
[36] E. Cambria, D. Olsher, and D. Rajagopal, "SenticNet 3: A common and common-sense knowledge base for cognition-driven sentiment analysis," in *Proc. AAAI*, Quebec, 2014, pp. 1515–1521.
[37] S. Poria, E. Cambria, A. Gelbukh, F. Bisio, and A. Hussain, "Sentiment data flow analysis by means of dynamic linguistic patterns," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 26–36, 2015.
[38] D. Tang, F. Wei, B. Qin, T. Liu, and M. Zhou, "Coooolll: A deep learning system for twitter sentiment classification," in *Proc. 8th Int. Workshop on Semantic Evaluation*, 2014.
[39] C. N. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proc. Int. Conf. Computational Linguistics*, 2014.
[40] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, "Learning sentiment-specific word embedding for Twitter sentiment classification," in *Proc. Annu. Meeting of the Association for Computational Linguistics*, 2014.
[41] S. Poria, E. Cambria, and A. Gelbukh, "Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis," in *Proc. EMNLP*, 2015, pp. 2539–2544.
[42] V. Cherkassky, "The nature of statistical learning theory," *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1564–1564, 1997.
[43] S. Madden, "From databases to big data," *IEEE Internet Comput.*, no. 3, pp. 4–6, 2012.
[44] A. G. Shoro and T. R. Soomro, "Big data analysis: Apache spark perspective," *Global J. Comp. Sci. Technol.*, vol. 15, no. 1, 2015.
[45] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?," *Neural Computat.*, vol. 16, no. 5, pp. 1063–1076, 2004.
[46] L. Devroye, L. Györfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. New York: Springer, 1996.
[47] E. Cambria and G. B. Huang, "Extreme learning machines," *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 30–59, 2013.
[48] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
[49] S. Ridella, S. Rovetta, and R. Zunino, "Circular backpropagation networks for classification," *IEEE Trans. Neural Networks*, vol. 8, no. 1, pp. 84–97, 1997.
[50] G. B. Huang, Q. Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *Proc. IEEE Int. Joint Conf. Neural Networks*, 2004.
[51] G. B. Huang, L. Chen, and C. K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.
[52] F. Bisio, P. Gastaldo, R. Zunino, and E. Cambria, "A learning scheme based on similarity functions for affective common-sense reasoning," in *Proc. IJCNN*, 2015, pp. 2476–2481.
[53] G. B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man Cybern. B*, vol. 42, no. 2, pp. 513–529, 2012.
[54] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 2013.
[55] H. Furuta, T. Kameda, Y. Fukuda, and D. M. Frangopol, "Life-cycle cost analysis for infrastructure systems: Life cycle cost vs. safety level vs. service life," in *Proc. Life-Cycle Performance of Deteriorating Structures: Assessment, Design and Management*, 2004.
[56] A. G. Carlyle, S. L. Harrell, and P. M. Smith, "Cost-effective HPC: The community or the cloud?" in *Proc. IEEE Int. Conf. Cloud Computing Technology and Science*, 2010.
[57] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark*. Sebastopol, CA: O'Reilly Media, 2015.
[58] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*. Philadelphia, PA: SIAM, 2005.
[59] R. E. Megginson, *An Introduction to Banach Space Theory*. New York: Springer, 2012, vol. 183.
[60] H. Zou, T. Hastie, and R. Tibshirani, "On the degrees of freedom of the lasso," *Ann. Stat.*, vol. 35, no. 5, pp. 2173–2192, 2007.
[61] C. De Mol, E. De Vito, and L. Rosasco, "Elastic-net regularization in learning theory," *J. Complex.*, vol. 25, no. 2, pp. 201–230, 2009.
[62] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley-Interscience, 1998.
[63] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least-squares classification," *Nato Sci. Ser. Sub Ser. III Comp. Syst. Sci.*, vol. 190, pp. 131–154, 2003.
[64] J. A. K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, vol. 9, no. 3, pp. 293–300, 1999.
[65] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learn. Res.*, vol. 9, pp. 1871–1874, 2008.
[66] R. Caruana, S. Lawrence, and G. Lee, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. Neural Information Processing Systems*, 2001.
[67] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.
[68] S. Gopalani and R. Arora, "Comparing apache spark and map reduce with performance analysis using k-means," *Int. J. Comp. Appl.*, vol. 113, no. 1, 2015.
[69] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Stat. Survey.*, vol. 4, no. 2010, pp. 40–79, 2010.
[70] D. A. McAllester, "Pac-bayesian stochastic model selection," *Machine Learn.*, vol. 51, no. 1, pp. 5–21, 2003.
[71] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. Int. Joint Conf. Artificial Intelligence*, San Francisco, CA, 1995, pp. 1137–1143.
[72] I. Guyon, A. Saffari, G. Dror, and G. Cawley, "Model selection: Beyond the Bayesian/frequentist divide," *J. Machine Learn. Res.*, vol. 11, pp. 61–87, Mar. 2010.
[73] D. Anguita, A. Ghio, L. Oneto, and S. Ridella, "In-sample model selection for support vector machines," in *Proc. Int. Joint Conf. Neural Networks*, San Jose, CA, 2011, pp. 1154–1161.
[74] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. London, UK: Chapman & Hall, 1993.
[75] L. Oneto, A. Ghio, S. Ridella, and D. Anguita, "Global rademacher complexity bounds: From slow to fast convergence rates," *Neural Processing Letter*, vol. 43, no. 2, pp. 567–602, Apr. 2016.
[76] L. Oneto, A. Ghio, S. Ridella, and D. Anguita, "Local rademacher complexity: Sharper risk bounds with and without unlabeled samples," *Neural Networks,* vol. 65, pp. 115-125, May 2015.
[77] D. A. McAllester, "Some pac-bayesian theorems," in *Proc. ACM Computational Learning Theory*, New York, 1998, pp. 230–234.
[78] G. Lever, F. Laviolette, and J. Shawe-Taylor, "Tighter pac-bayes bounds through distribution-dependent priors," *Theoretic. Comp. Sci.*, vol. 473, pp. 4–28, 2013.
[79] S. Floyd and M. Warmuth, "Sample compression, learnability, and the vapnik-chervonenkis dimension," *Machine Learn.*, vol. 21, no. 3, pp. 269–304, 1995.
[80] T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi, "General conditions for predictivity in learning theory," *Nature*, vol. 428, no. 6981, pp. 419–422, 2004.
[81] A. Inoue and L. Kilian, "In-sample or out-of-sample tests of predictability: Which one should we use?" *Econometric Rev.*, vol. 23, no. 4, pp. 371–402, 2005.
[82] F. Cheng, J. Yu, and H. Xiong, "Facial expression recognition in jaffe dataset based on gaussian process classification," *IEEE Trans. Neural Networks*, vol. 21, no. 10, pp. 1685–1690, 2010.
[83] D. Anguita, A. Ghio, S. Ridella, and D. Sterpi, "K-fold cross validation for error rate estimate in support vector machines," in *Proc. Int. Conf. Data Mining*, 2009.
[84] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Am. Stat. Assoc.*, vol. 58, no. 301, pp. 13–30, 1963.
[85] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," *J. Machine Learn. Res.*, vol. 3, pp. 463–482, Mar. 2003.
[86] A. Rakhlin, S. Mukherjee, and T. Poggio, "Stability results in learning theory," *Analysis Appl*, vol. 3, no. 4, pp. 397–417, 2005.
[87] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, "A scalable bootstrap for massive data," *J. R. Stat. Soc. B Stat. Methodol.*, vol. 76, no. 4, pp. 795–816, 2014.
[88] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, "The big data bootstrap," in *Proc. Int. Conf. Machine Learning*, 2012.
[89] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan, "Bootstrapping big data," in *Proc. Advances in Neural Information Processing Systems, Workshop: Big Learning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.
[90] L. Oneto, B. Pilarz, A. Ghio, and D. Anguita, "Model selection for big data: Algorithmic stability and bag of little bootstraps on gpus," in *Proc. Eur. Symp. Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2015.
[91] R. Dietrich, M. Opper, and H. Sompolinsky, "Statistical mechanics of support vector networks," *Phys. Rev. Lett.*, vol. 82, no. 14, pp. 2975, 1999.
[92] E. Cambria, "Affective computing and sentiment analysis," *IEEE Intell. Syst.*, vol. 31, no. 2, pp. 102–107, 2016.
[93] E. Cambria, P. Gastaldo, F. Bisio, and R. Zunino, "An ELM-based model for affective analogical reasoning," *Neurocomputing*, vol. 149, no. A, pp. 443–455, Feb. 2015.
[94] E. Cambria, J. Fu, F. Bisio, and S. Poria, "AffectiveSpace 2: Enabling affective intuition for concept-level sentiment analysis," in *Proc. AAAI*, Austin, TX, 2015, pp. 508–514.
[95] E. Cambria and A. Hussain, *Sentic Computing: A Common-Sense-Based Framework for Concept-Level Sentiment Analysis*. Cham, Switzerland: Springer, 2015.
[96] G. Huang, E. Cambria, K. Toh, B. Widrow, and Z. Xu, "New trends of learning in computational intelligence [guest editorial]," *IEEE Comput. Intell. Mag.*, vol. 10, no. 2, pp. 16–17, 2015.