Computer Science Review 37 (2020) 100254

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Julia language in machine learning: Algorithms, applications, and open issues

Kaifeng Gao [a], Gang Mei [a,*], Francesco Piccialli [b,*], Salvatore Cuomo [b,*], Jingzhi Tu [a], Zenan Huo [a]

[a] School of Engineering and Technology, China University of Geosciences (Beijing), 100083, Beijing, China
[b] Department of Mathematics and Applications R. Caccioppoli, University of Naples Federico II, Naples, Italy

ARTICLE INFO

Article history:
Received 27 February 2020
Received in revised form 25 April 2020
Accepted 6 May 2020
Available online 16 May 2020

Keywords:
Julia language
Machine learning
Supervised learning
Unsupervised learning
Deep learning
Artificial neural networks

ABSTRACT

Machine learning is driving development across many fields in science and engineering. A simple and efficient programming language could accelerate applications of machine learning in various fields. Currently, the programming languages most commonly used to develop machine learning algorithms include Python, MATLAB, and C/C ++. However, none of these languages well balance both efficiency and simplicity. The Julia language is a fast, easy-to-use, and open-source programming language that was originally designed for high-performance computing, which can well balance the efficiency and simplicity. This paper summarizes the related research work and developments in the applications of the Julia language in machine learning. It first surveys the popular machine learning algorithms that are developed in the Julia language. Then, it investigates applications of the machine learning algorithms implemented with the Julia language. Finally, it discusses the open issues and the potential future directions that arise in the use of the Julia language in machine learning.

Contents

* Corresponding authors.
E-mail addresses: gang.mei@cugb.edu.cn (G. Mei), francesco.piccialli@unina.it (F. Piccialli), salvatore.cuomo@unina.it (S. Cuomo).

https://doi.org/10.1016/j.cosrev.2020.100254

List of Abbreviations

AD	Algorithmic Differentiation
APIs	Application Programming Interfaces
CNN	Convolutional Neural Network
DPMM	Dirichlet Process Mixture Model
ELM	Extreme Learning Machine
FFGs	Forney-style Factor Graphs
GBDT	Gradient-Boosting Decision Tree
GMMs	Gaussian Mixture Models
GPU	Graphics Processing Unit
ICA	Independent Component Analysis
IoT	Internet of Things
JIT	Just-In-Time
*k*NN	*k*-Nearest Neighbors
LLVM	Low-Level Virtual Machine
NLP	Natural Language Processing
ODPS	Open Data Processing Service
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
SVD	Singular Value Decomposition
SVM	Support Vector Machine

1. Introduction

Machine learning is currently one of the most rapidly growing technical fields, lying at the intersection of computer science and statistics and at the core of artificial intelligence and data science [1–4]. Machine learning technology powers many aspects of modern society, from web searches to content filtering on social networks to recommendations on electronic commerce websites. Recent advances in machine learning methods promise powerful new tools for practicing scientists. Modern machine learning methods are closely related to scientific application [5,6]; see Fig. 1.

Python, MATLAB, Go, R, and C/C++ are widely used programming languages in machine learning. Python has proven to be a very effective programming language and is used in many scientific computing applications [7]. MATLAB combines the functions of numerical analysis, matrix calculation, and scientific data visualization in an easy-to-use manner. Both Python and MATLAB are "Plug-and-Play" programming languages; the algorithms are prepackaged and mostly do not require learning processes, but they are used to solve large-scale tasks at a slow speed and have very strict requirements for memory and computing power [8]. In addition, MATLAB is commercial.

Go is an open-source programming language that makes it easy to build simple, reliable, and efficient software. Go is syntactically similar to C, but with memory safety, garbage collection, and structural typing. Rather than call out to libraries written in other languages, developers can work with machine learning libraries written directly in Go. However, the current machine learning libraries written in Go are not extensive. R is a language and environment for statistical computing and graphics. R provides a wide variety of statistical and graphical techniques, and is highly extensible. One of the advantages of R is that it can easily produce high-quality drawings. However, R stores data in system memory (RAM), which is a constraint when analyzing big data.

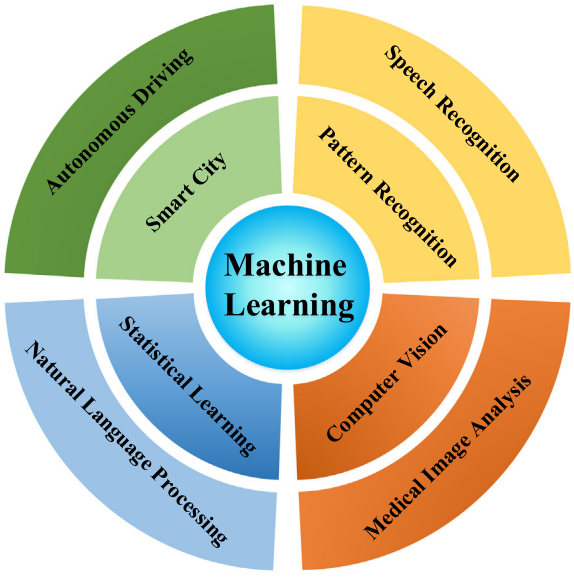

Fig. 1. Main applications of machine learning.

C/C++ is one of the main programming languages in machine learning. It is of high efficiency and strong portability. However, the development and implementation of machine learning algorithms with C/C++ is not easy due to the difficulties in learning and using C/C++. In machine learning, the availability of large data sets is increasing, and the demand for general large-scale parallel analysis tools is also increasing [9]. Therefore, it is necessary to choose a programming language with both simplicity and good performance.

Julia is a simple, fast, and open-source language [10]. The efficiency of Julia is almost comparable to that of static programming languages such as C/C++ and Fortran [11]. Julia is rapidly becoming a highly competitive language in data science and general scientific computing. Julia is as easy to use as R, Python, and MATLAB.

Julia was originally designed for high-performance scientific computing and data analysis. Julia can call many other mature high-performance basic codes, such as linear algebra and fast Fourier transforms. Similarly, Julia can call C++ language functions directly without packaging or special application programming interfaces (APIs). In addition, Julia has special designs for parallel computing and distributed computing. In high-dimensional computing, Julia has more advantages than C++ [9]. In the field of machine learning, Julia has developed many third-party libraries, including some for machine learning.

In this paper, we systematically review and summarize the development of the Julia programming language in the field of machine learning by focusing on the following three aspects:

(1) Machine learning algorithms developed in the Julia language.

(2) Applications of the machine learning algorithms implemented with the Julia language.

(3) Open issues that arise in the use of the Julia language in machine learning.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to the Julia language. Section 3 summarizes

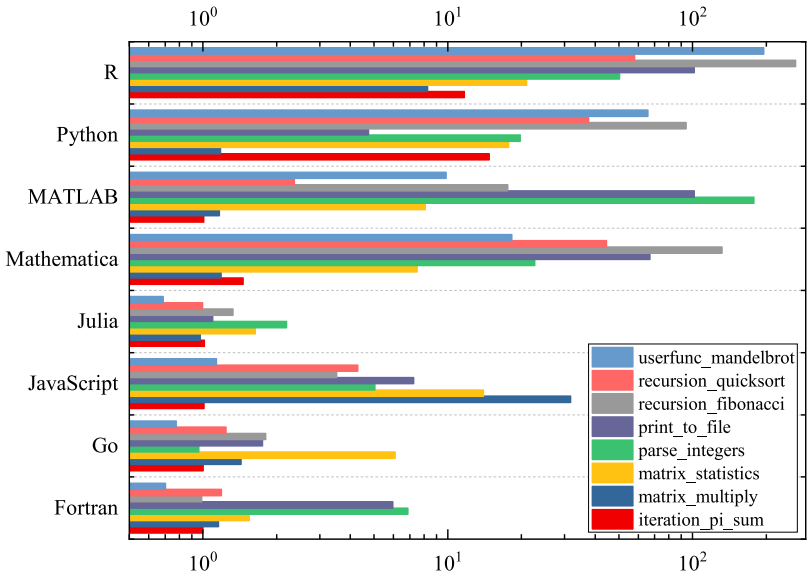

Fig. 2. Julia benchmarks (the benchmark data shown above were computed with Julia v1.0.0, Go 1.9, Javascript V8 6.2.414.54, MATLAB R2018a, Anaconda Python 3.6.3, and R 3.5.0. C and Fortran are compiled with gcc 7.3.1, taking the best timing from all optimization levels. C performance = 1.0, smaller is better [12]).

the machine learning algorithms developed in Julia language. Section 4 introduces applications of the machine learning algorithms implemented with Julia language. Section 5 presents open issues occurring in the use of Julia language in machine learning. Finally, Section 6 concludes this survey.

2. A brief introduction to the Julia language

Julia is a modern, expressive, and high-performance programming language for scientific computing and data processing. Its development started in 2009, and the current stable release as of April 2020 is v1.4.0. Although this low version number indicates that the language is still developing rapidly, it is stable enough to enable the development of research code. Julia's grammar is as readable as that of MATLAB or Python, and it can approach the C/C++ language in performance by compiling in real time. In addition, Julia is a free, open-source language that runs on all popular operation systems.

With the low-level virtual machine (LLVM)-based just-in-time (JIT) compiler, Julia provides powerful computing performance. [13,14]; see Fig. 2. Julia also incorporates some important features from the beginning of its design, such as excellent support for parallelism [15] and a practical functional programming orientation, which were not fully implemented in the development of scientific computing languages decades ago. Julia can also be embedded in other programming languages. These advantages make Julia a universal language.

Julia successfully combines the high performance of a static programming language with the flexibility of a dynamic programming language [14]. It provides built-in primitives for parallel computing at every level: instruction level parallelism, multithreading and distributed computing. The Julia modules allow users to suspend and resume computations with full control of communication without having to manually interface with the operating system's scheduler. Besides, Julia provides a multiprocessing environment based on message passing to allow programs to run on multiple processes in separate memory domains at once [16]. Moreover, the use of the high-level Julia programming language enables new and dynamic approaches for graphics processing unit (GPU) programming, and Julia GPU code can be highly generic and flexible, without sacrificing performance [15,17]. However, parallel computing has not yet reached the required level of richness and interactivity [10]. The Julia language could further improve the efficiency of data division and combination, and optimize the parallel algorithms.

3. Julia in machine learning: Algorithms

3.1. Overview

This section describes machine learning algorithm packages and toolkits written either in or for Julia. Most applications of machine learning algorithms in Julia can be divided into supervised learning and unsupervised learning algorithms. However, more complex algorithms, such as deep learning, artificial neural networks, and extreme learning machines, include both supervised learning and unsupervised learning, and these require separate classification; see Fig. 3.

Supervised learning learns the training samples with class labels and then predicts the classes of data outside the training samples. All the markers in supervised learning are known; therefore, the training samples have low ambiguity. Unsupervised learning learns the training samples without class labels to discover the structural knowledge in the training sample set. All categories in unsupervised learning are unknown; thus, the training samples are highly ambiguous.

3.2. Supervised learning algorithms developed in Julia

Supervised learning infers a model from labeled training data. Supervised learning algorithms developed in Julia mainly include classification and regression algorithms; see Fig. 4.

Bayesian model

There are two key points in the definition of Bayesian model: independence between features and the Bayesian theorem. One of the most important research areas of Bayesian model is Bayesian linear regression. Bayesian linear regression solves the problem of overfitting in maximum likelihood estimation. Moreover, it makes full use of data samples and is suitable for modeling complex data [18,19]. In addition to regression, Bayesian reasoning can also be applied in other fields. Some researchers have conducted research on naive Bayes in image recognition and text classification.

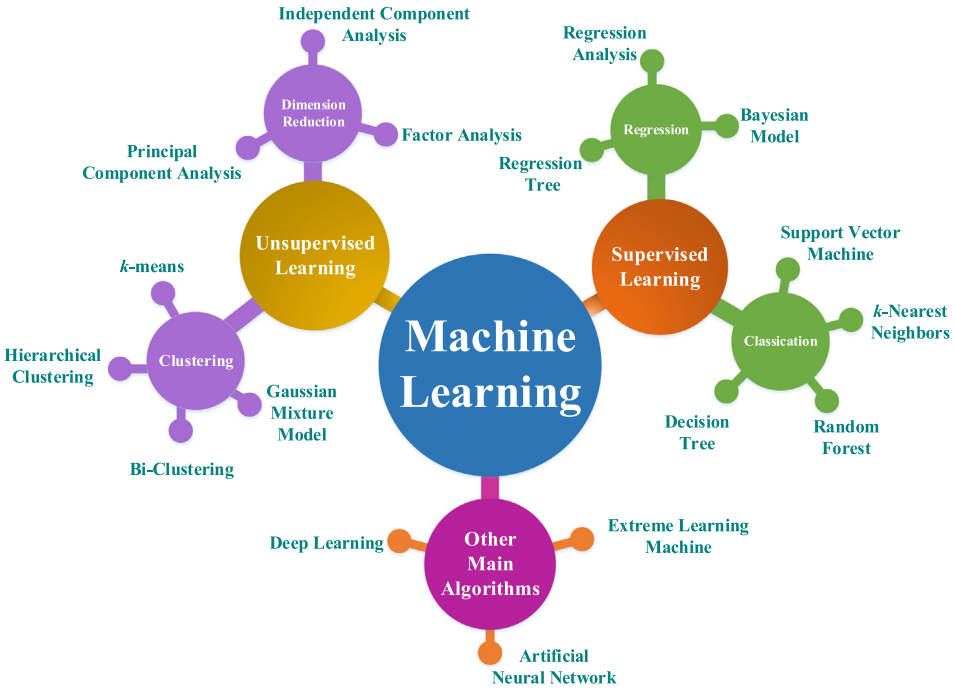

Fig. 3. Main machine learning algorithms.

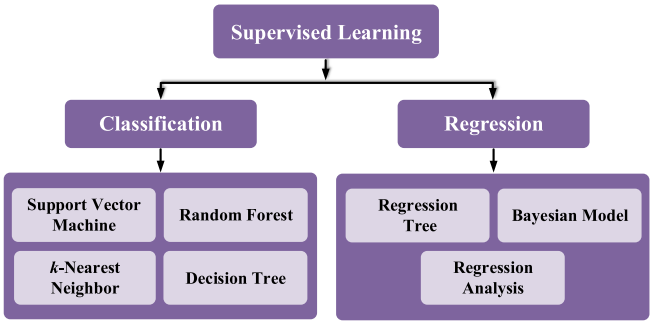

Fig. 4. Main supervised learning algorithms developed in Julia.

There are some Bayesian model packages and algorithms developed in mature languages. Strickland et al. [20] developed the Python package **Pyssm**, which was developed for time series analysis using a linear Gaussian state-space model. Mertens et al. [21] developed a user-friendly Python package **Abrox** for approximate Bayesian computation with a focus on model comparison. There are also Python packages **BAMSE** [22], **BayesPy** [23], **PyMC** [24] and so on. Moreover, Vanhatalo et al. [25] developed the MATLAB toolbox **GPstuff** for Bayesian modeling with Gaussian processes, and Zhang et al. [26] developed the MATLAB toolbox **BSmac**, which implements a Bayesian spatial model for brain activation and connectivity.

The Julia language is also used to develop packages for the Bayesian model. **Gen** [27] is a probabilistic programming language proposed by Cusumano and Mansinghka that can be embedded in Julia. This language provides a structure for the optimization of the automatic generation of custom reasoning strategies for static analysis based on an objective probability model. They described **Gen**'s language design informally and used an example Bayesian statistical model for robust regression to show that **Gen** is more expressive than **Stan**, a widely used language for hierarchical Bayesian modeling. Cox et al. [28] explored a specific probabilistic programming paradigm, namely, message passing in Forney-style factor graphs (FFGs), in the context of the automated design of efficient Bayesian signal processing algorithms. Moreover, they developed **ForneyLab.jl** as a Julia Toolbox for message passing-based inference in FFGs.

Due to the increasing availability of large data sets, the need for a general-purpose massively parallel analysis tool is becoming ever greater. Bayesian nonparametric mixture models, exemplified by the Dirichlet process mixture model (DPMM), provide a principled Bayesian approach to adapt model complexity to the data. Dinari et al. [9] used Julia to implement efficient and easily modifiable distributed inference in DPMMs.

K -nearest neighbors (kNN)

The *k*NN algorithm has been widely used in data mining and machine learning due to its simple implementation and distinguished performance. A training data set with a known label category is used, and for a new data set, the *k* instances closest to the new data are found in the feature space of the training data set. If most of the instances belong to a category, the new data set belongs to this category.

At present, there are many packages developed for the *k*NN algorithm in the Python language. Among these, **scikit-learn** and **Pypl** are the most commonly used packages. It should be noted that **scikit-learn** and **Pypl** are not specially developed for the *k*NN algorithm; they contain many other machine learning algorithms. In addition, Bergstra et al. [29] developed **Hyperopt** to define a search space that encompasses many standard components and common patterns of composing them.

Julia is also used to develop packages for the *k*NN algorithm. **NearestNeighbors.jl** [30] is a package written in Julia to perform high-performance nearest neighbor searches in arbitrarily high dimensions. This package can realize *k*NN searches and range searches.

Decision tree, regression tree, and random forest

Mathematically, a decision tree is a graph that evaluates a limited number of probabilities to determine a reliable classification for each data point. A regression tree is the opposite of a

decision tree and is suitable for solving regression problems. It does not predict labels but predicts a continuous change value. Random forests are a set of decision trees or regression trees that work together [31]. The set of decision trees (or continuous y regression trees) is constructed by performing bootstrapping on the data sets and averaging or acquiring pattern prediction (called "bagging") from the trees. Subsampling of features is used to reduce generalization errors [32]. An ancillary result of the bootstrapping procedure is that the data not sampled in each bootstrap (called "out-of-bag" data) can be used to estimate the generalization error as an alternative to cross-validation [33].

Many packages have been developed for decision trees, regression trees, and random forests. For example, the above three algorithms are implemented in **Spark2 ML** and **scikit-learn** using Python. In addition, Upadhyay et al. [34] proposed land-use and land-cover classification technology based on decision trees and *k*-nearest neighbors, and the proposed techniques are implemented using the **scikit-learn** data mining package for python. Keck [35] proposed a speed-optimized and cache-friendly implementation for multivariate classification called **FastBDT**, which provides interfaces to C/C++, Python, and TMVA. Yang et al. [36] used the open data processing service (ODPS) and Python to implement the gradient-boosting decision tree (GBDT) model.

DecisionTree.jl [37], written in the Julia language, is a powerful package that can realize decision tree, regression tree, and random forest algorithms very well. The package has two functions, and the ingenious use of these functions can help us realize these three algorithms.

Support vector machine (SVM)

In SVM, the objective is to find a hyperplane in high-dimensional space, which represents the maximum margin between any two instances of two types of training data points (support vectors) or maximizes the correlation function when it cannot be separated. The so-called kernel similarity function is used to design the non-linear SVM [38].

Currently, there are textbook style implementations of two popular linear SVM algorithms: Pegasos [39], Dual Coordinate Descent. **LIBSVM** developed by the Information Engineering Institute of Taiwan University is the most widely used SVM tool [40]. **LIBSVM** includes standard SVM algorithm, probability output, support vector regression, multi-classification SVM and other functions. Its source code is originally written by C. It provides Java, Python, R, MATLAB, and other language invocation interfaces.

SVM.jl [41], **MLJ.jl** [42], and **LIBSVM.jl** [43] are native Julia implementations of SVM algorithm. However, **LIBSVM.jl** is more comprehensive than **SVM.jl**. **LIBSVM.jl** supports all libsvm models: classification c-svc, nu-svc, regression: epsilon-svr, nu-svr and distribution estimation: a class of support vector machines and **ScikitLearn.jl** [44] API. In addition, the model object is represented by a support vector machine of Julia type. The SVM can easily access the model features and can be saved as a JLD file.

Regression analysis

Regression analysis is an important supervised learning algorithm in machine learning. It is a predictive modeling technique, which constructs the optimal solution to estimate unknown data through the sample and weight calculation. Regression analysis is widely used in the fields of the stock market and medical data analysis.

Python has been widely used to develop a variety of third-party packages for regression analysis, including **scikit-learn** and **orange**. The **scikit-learn** package is a powerful Python module, which supports mainstream machine learning algorithms such as regression, clustering, classification and neural network [45–47].

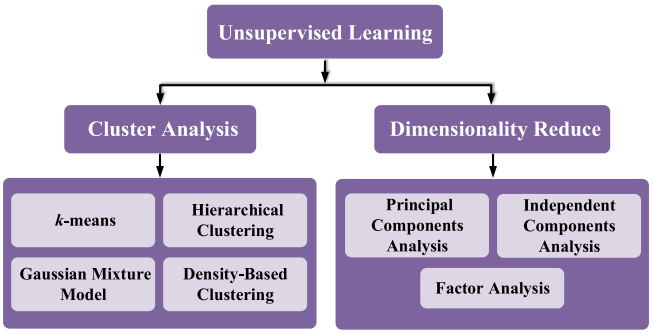

Fig. 5. Main unsupervised learning algorithms developed in Julia.

The **orange** package is a component-based data mining software, which can be used as a module of Python programming language, especially suitable for classification, clustering, regression and other work [48,49]. MATLAB also supports the regression algorithm. By invoking commands such as *regress* and *stepwise* in the statistical toolbox of MATLAB, regression operation can be performed conveniently on the computer.

The Julia language is also used to develop a package, **Regression.jl** [50], to perform the regression analysis. The **Regression.jl** package seeks to minimize empirical risk based on **EmpiricalRisk.jl** [51] and provides a set of algorithms for performing regression analysis. It supports multiple linear regression, nonlinear regression, and other regression algorithms. In addition, the **Regression.jl** package also provides a variety of solvers such as analytical solution (for linear and ridge regression) and gradient descent.

3.3. Unsupervised learning algorithms developed in Julia

Unsupervised learning is a type of self-organized learning that can help find previously unknown patterns in a dataset without the need for pre-existing labels. Two of the main methods used in unsupervised learning are dimensionality reduction and cluster analysis; see Fig. 5.

Gaussian mixture models (GMMs)

GMMs are probabilistic models for representing normally distributed subpopulations within an overall population. GMMs use Gaussian distribution as the basic parameter model, accurately characterizes the data distribution by combining multiple Gaussian distributions, and use the expectation–maximization algorithm for training. Compared with the Gaussian models, the GMMs provide greater flexibility and precision in modeling the underlying statistics of sample data [52]. Generally, the GMMs are used to solve problems such as image segmentation and dynamic target detection [53].

Currently, there are many libraries that can implement Gaussian mixture models; these include packages developed with Python, such as **PyBGMM** and **numpy-ml**, and packages developed with C++, such as **Armadillo**. There are also some GMM packages for specialized fields. Bruneau et al. [54] proposed a new Python package for nucleotide sequence clustering, which implements a Gaussian mixture model for DNA clustering. Holoien et al. [55] developed a new open-source tool, **EmpiriciSN**, written in Python, for performing extreme deconvolution Gaussian mixture modeling.

To the best of the authors' knowledge, there is no mature Julia package for the GMMs. **GmmFlow.jl** [56] is a Julia library that can implement some simple functions of the GMMs, including

model generation and cluster mapping. But the algorithm optimization could be improved. **GaussianMixtures.jl** [57] is another Julia package for the GMMs. This package has implemented both diagonal covariance and full covariance GMMs, and full covariance variational Bayes GMMs. However, the package is slightly strict with data types. **ScikitLearn.jl** implements the popular **scikit-learn** interface and algorithms in Julia, and it can access approximately 150 Julia and Python models, including the Gaussian mixture model. Moreover, Srajer et al. [58] used algorithmic differentiation (AD) tools in a GMM fitting algorithm.

K -means

The *k*-means clustering algorithm is an iterative clustering algorithm. It first randomly selects *k* objects as the initial clustering center, then calculates the distance between each object and each seed clustering center, and assigns each object to the nearest clustering center. Cluster centers and the objects assigned to them represent a cluster. As an unsupervised clustering algorithm, *k*-means is widely used because of its simplicity and effectiveness.

The *k*-means algorithm is a classic clustering method, and many programming languages have developed packages related to it. The third-party package **scikit-learn** in Python implements the *k*-means algorithm [47,59]. The *Kmeans* function in MATLAB can also implement a *k*-means algorithm [60]. In addition, many researchers have implemented the *k*-means algorithm in the C/C++ programming language.

Julia has also been used to develop a specific package, **Clustering.jl** [61], for clustering. **Clustering.jl** provides several functions for data clustering and clustering quality evaluation. Because **Clustering.jl** has comprehensive and powerful functions, this package is a good choice for *k*-means.

Hierarchical clustering

Hierarchical clustering is a kind of clustering algorithm that performs clustering by calculating the similarity between data points of different categories [62–64]. The strategy of cohesive hierarchical clustering is to first treat each object as a cluster and then merge these clusters into larger and larger clusters until all objects are in one cluster or some termination condition is satisfied.

The commonly used Python packages for hierarchical clustering are **scikit-learn** and **scipy**. Hierarchical clustering within the **scikit-learn** package is implemented in the *sklearn.cluster* method, which includes three important parameters: the number of clusters, the connection method, and connection measurement options [47]. **scipy** implements hierarchical clustering with the *scipy.cluster* method [65]. In addition, programming languages such as MATLAB and C/C++ can also perform hierarchical clustering [66].

The package **QuickShiftClustering.jl** [67], written using Julia, can realize hierarchical clustering algorithms. This package is quite easy to use. It provides three functions: clustering matrix data, clustering labels, and creating hierarchical links to achieve hierarchical clustering [68].

Bi-clustering

Bi-Clustering algorithm is based on traditional clustering. Its basic idea is to cluster rows and columns of matrices through traditional clustering, and then merge the clustering results. Bi-Clustering algorithm solves the bottleneck problem of traditional clustering in high-dimensional data. Data sets in reality are mostly high-dimensional and inherently sparse. Traditional clustering algorithms often fail to detect meaningful clustering in high-dimensional data sets. However, Bi-Clustering can detect clusters of any shape and position in space, and it is an effective method to solve the problem of subspace clustering in high-dimensional data sets [69,70]. To search for local information better in the data matrix, researchers put forward the concept of bi-clustering.

The package **scikit-learn** can implement bi-clustering, and the implementation module is *sklearn.cluster.bicluster*. At present, bi-clustering is mainly applied to highthroughput detection technologies such as gene chips and DNA microarrays.

The Julia language is also used to develop packages that implement bi-clustering. For example, **Kpax3** [71] is a Bayesian method for multi-cluster multi-sequence alignment. Bezanson et al. [10] used a Bayesian dual clustering model, which extended and improved the model originally introduced by Pessia et al. [71]. They wrote the **kpax3.jl** library package in Julia and the output contains multiple text files containing a cluster of rows and columns of the input dataset.

Principal component analysis (PCA)

PCA is a method of statistical analysis and a simplified data set. It uses an orthogonal transformation to linearly transform observations of a series of possibly related variables and then project them into a series of linearly uncorrelated variables. These uncorrelated variables are called principal components. PCA is often used to reduce the dimensionality of a data set while maintaining the features that have the largest variance contribution in the data set.

Python is the most frequently used language for developing PCA algorithms. The **scikit-learn** package provides a class, *sklearn.decomposition.PCA* [72], to implement PCA algorithms in the *sklearn.decomposition* module. Generally, the PCA class does not need to adjust parameters very much but needs to specify the target dimension or the variance of the principal components after dimensionality reduction. In addition, many researchers have developed related application packages using the C++ programming language. These include the **ALGLIB** [73] package and the class **cv :: PCA** [74] in OpenCV.

To the best of the authors' knowledge, there is no mature Julia package specifically for PCA. However, **MultivariateStats.jl** [75] is a Julia package for multivariate statistics and data analysis. This package defines a PCA type to represent a PCA model and provides a set of methods to access properties.

Independent component analysis (ICA)

ICA is a new signal processing technology developed in recent years. The ICA method is based on mutual statistical independence between sources. In the practical applications of signal processing, especially in communication and biomedicine, it is important to eliminate noise data. Traditional signal processing techniques for noise cancellation include band-pass filtering, fast Fourier transform, autocorrelation, autoregressive modeling, adaptive filtering, Kalman filtering and singular value decomposition. The traditional filtering technology is based on the assumption that noise is the only additive, which is not suitable for multi-sensor observation of mixed signals [76]. However, the ICA algorithm has strong robustness to additive noise, and it is one of the most promising methods to solve the problem of blind noise suppression [77,78]. Moreover, in contrast to traditional signal separation methods based on feature analysis, such as singular value decomposition (SVD) and PCA, ICA is an analysis method based on higher-order statistical characteristics. In many applications, the analysis of higher-order statistical characteristics is more practical.

Python is the most frequently used language in developing ICA algorithms. The **scikit-learn** package has developed a class, *FastICA* [79], to implement ICA algorithms in the *sklearn.decomposition* module. In addition, Brian Moore [80] developed a

PCA and ICA Package using the MATLAB programming language. The PCA and ICA algorithms are implemented as functions in this package, and it includes multiple examples to demonstrate their usage.

To the best of the authors' knowledge, ICA does not have a mature software package developed in the Julia language. However, **MultivariateStats.jl** [75], like a Julia package for multivariate statistics and data analysis, defines an ICA type representing the ICA model and provides a set of methods to access the attributes.

3.4. Other main algorithms

In addition to supervised learning algorithms and unsupervised learning algorithms, machine learning algorithms include a class of algorithms that are more complex and cannot be categorized into a specific category. For example, artificial neural networks can implement supervised learning, unsupervised learning, reinforcement learning, and self-learning. Deep learning algorithms are based on artificial neural network algorithms and can perform supervised learning, unsupervised learning, and semisupervised learning. Extreme learning machines were proposed for supervised learning algorithms but were extended to unsupervised learning in subsequent developments.

Deep learning

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [5]. Several deep learning frameworks, such as the depth neural network, the convolutional neural network, the depth confidence network and the recursive neural network, have been applied to computer vision, speech recognition, natural language processing, image recognition, and bioinformatics and have achieved excellent results.

It has been several years since the birth of deep learning algorithms. Many researchers have improved and developed deep learning algorithms. Python is the most frequently used language in developing deep learning algorithms. For example, **PyTorch** [81,82] and **ALiPy** [83] are Python packages with many deep learning algorithms. Moreover, Tang et al. developed **GCNv2** [84] using C++ and Python, Huang et al. wrote **Mask Scoring R-CNN** [85] using Python, Hanson and Frazier-Logue compared the Dropout [86] algorithm with the SDR [87] algorithm, and Luo et al. [88] proposed and used Python to write AdaBound (a new adaptive optimization algorithm).

Julia has also been used to develop various deep learning algorithms. For example, AD allows the exact computation of derivatives given only an implementation of an objective function, and Srajer et al. [58] wrote an AD tool and used it in a hand-tracking algorithm.

Augmentor is a software package available in both Python and Julia that provides a high-level API for the expansion of image data using a stochastic, pipeline-based approach that effectively allows images to be sampled from a distribution of augmented images at runtime [89]. To demonstrate the API and to highlight the effectiveness of augmentation on a well-known dataset, a short experiment was performed. In the experiment, the package is used on a convolutional neural network (CNN) [90].

MXNet.jl [91], **Knet.jl** [92], **Flux.jl** [93], and **TensorFlow.jl** [94] are deep learning frameworks with both efficiency and flexibility. At its core, **MXNet.jl** contains a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. **MXNet.jl** is portable and lightweight, scaling effectively to multiple GPUs and multiple machines.

Artificial neural networks

A neural network is a feedforward network consisting of nodes ("neurons"), each side of which has weights. These allow the network to form a mapping between the input and output [95]. Each neuron that receives input from a previous neuron consists of the following components: the activation, a threshold, the time at which the newly activated activation function is calculated and the output function of the activation output.

At present, the framework of a neural network model is usually developed in C++ or Python. **DLL** is a machine learning framework written in C++ [96]. It supports a variety of neural network layers and standard backpropagation algorithms. It can train artificial neural networks and CNNs and support basic learning options such as momentum and weight attenuation. **scikit-learn**, a machine learning library based on Python, also supports neural network models [47].

Employing the Julia language, **Diffiqflux.jl** [97] is a package that integrates neural networks and differential equations. Rackauckas et al. [97] described differential equations from the perspective of data science and discuss the complementarity between machine learning models and differential equations. These authors demonstrated the ability to combine **DifferentialEquations.jl** [98] defined differential equations into **Flux**-defined neural networks. **Backpropneuralnet.jl** [58] is an easy-to-use neural network package.

Extreme learning machine (ELM)

ELM [99] is a variant of Single Hidden Layer Feedforward Networks (SLFNs). Because its weight is not adjusted iteratively, it deviates greatly. This significantly improves the efficiency when training the neural networks.

The basic algorithm of ELM and Multi-Layer [100]/Hierarchical [101] ELM have been implemented in **HP-ELM**. Meanwhile, C/C++, MATLAB, Python and JAVA versions are provided. **HP-ELM** includes GPU acceleration and memory optimization, which is suitable for large data processing. **HP-ELM** supports LOO (Leave One Out) and *k*-fold cross-validation to dynamically select the number of hidden layer nodes. The available feature maps include linear function, Sigmoid function, hyperbolic sinusoidal function, and three radial basis functions.

According to ELM, parameters of hidden nodes or neurons are not only independent of training data but also independent of each other. Standard feedforward neural networks with hidden nodes have universal approximation and separation capabilities. These hidden nodes and their related maps are terminologically ELM random nodes, ELM random neurons or ELM random features. Unlike traditional learning methods, which need to see training data before generating hidden nodes or neuron parameters, ELM could generate hidden nodes or neuron parameters randomly before seeing training data. **Elm.jl** [102] is an easy-to-use extreme learning machine package.

3.5. List of commonly used Julia packages

We summarize the commonly used Julia language packages and the machine learning algorithms that these packages primarily support; see the investigation in Table 1.

4. Julia in machine learning: Applications

4.1. Overview

Machine learning is one of the fastest-growing technical fields nowadays. It is a cross-cutting field of statistics and computer science [1–3]. Machine learning specializes in how computers simulate or implement human learning behaviors. By acquiring

Table 1
Commonly used Julia language packages.

Julia packages	Ref.	Supervised learning					Unsupervised learning						Other		
		Bayesian model	*k*NN	Random forest	SVM	Regression analysis	GMM	*k*-means	Hierarchical Clustering	Bi-Clustering	PCA	ICA	Deep Learning	ANN	ELM
ForneyLab.jl	[28]	✓													
NearestNeighbors.jl	[30]		✓												
DecisionTree.jl	[37]			✓											
SVM.jl	[41]				✓										
MLJ.jl	[42]	✓		✓	✓	✓									
LIBSVM.jl	[43]				✓										
ScikitLearn.jl	[44]				✓		✓								
Regression.jl	[50]					✓									
EmpiricalRisk.jl	[51]					✓									
Clustering.jl	[61]							✓							
QuickShiftClustering.jl	[67]								✓						
kpax3.jl	[8]									✓					
MultivariateStats.jl	[75]										✓	✓			
MXNet.jl	[91]												✓		
Knet.jl	[92]												✓	✓	
Flux.jl	[93]												✓	✓	✓
TensorFlow.jl	[94]												✓		
Diffiqflux.jl	[97]													✓	
DifferentialEquations.jl	[98]													✓	
Backpropneuralnet.jl	[58]													✓	
Elm.jl	[102]														✓
GmmFlow.jl	[56]						✓								
GaussianMixtures.jl	[57]						✓								

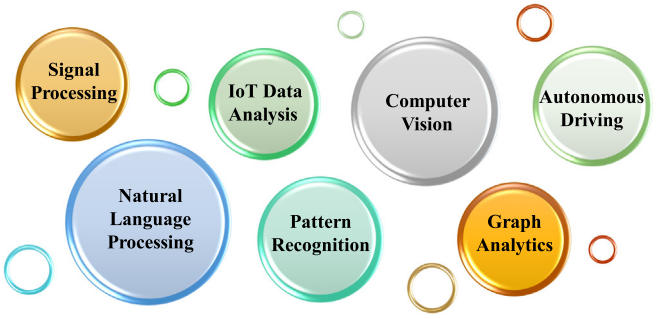

Fig. 6. Major applications of machine learning using Julia language.

new knowledge and skills, the existing knowledge structure is reorganized to improve its performance.

Julia, as a programming language with the rise of machine learning, has corresponding algorithmic library packages in most machine learning applications. In the following, we summarize the applications of Julia in machine learning. As shown in Fig. 6, the current applications of Julia programming language in machine learning mainly focus on the Internet of Things (IoT), computer vision, autonomous driving, pattern recognition, etc.

4.2. Analysis of IoT data

The IoT, also called the Internet of Everything or the Industrial Internet, is a new technology paradigm envisioned as a global network of machines and devices capable of interacting with each other [103]. The application of the IoT in industry, agriculture, the environment, transportation, logistics, security, and other infrastructure fields effectively promotes the intelligent development of these areas and more rationally uses and allocates limited resources, thus improving the efficiency of these fields [104–106]. Machine learning has brought enormous development opportunities for the IoT and has a significant impact on existing industries [107,108].

Invenia Technical Computing used the Julia language to expand its energy intelligence system [109]. They optimized the entire North American grid and used the energy intelligent system (EIS) and various signals to directly improve the day-ahead planning process. They used the latest research in machine learning, complex systems, risk analysis, and energy systems. In addition, Julia provided Invenia Technical Computing with versatility in terms of programming style, parallelism, and language interoperability [109].

Fugro Roames engineers [110] used the Julia language to implement machine learning algorithms to identify network faults and potential faults, achieving a 100-fold increase in speed. Protecting the grid means ensuring that all power lines, poles, and wires are in good repair, which used to be a laborious manual task that required thousands of hours to travel along the power line. Fugro Roames engineers have developed a more effective way to identify threats to wires, poles, and conductors. Using a combination of LiDAR and high-resolution aerial photography, they created a detailed three-dimensional map of the physical conditions of the grid and possible intrusions. Then, they used machine learning algorithms to identify points on the network that have failed or are at risk of failure [110].

4.3. Computer vision

Computer vision is a simulation of biological vision using computers and related equipment. Its main task is to obtain the three-dimensional information of the corresponding scene by processing collected pictures or videos. Computer vision includes image processing and pattern recognition. In addition, it also includes geometric modeling and recognition processes. The realization of image understanding is the ultimate goal of computer vision. Machine learning is developing, and computer vision research has gradually shifted from traditional models to deep learning models represented by CNNs and deep Boltzmann machines.

Computer vision is currently widely applied in the fields of biological and medical image analysis [111], urban streetscapes [37,112], rock type identification [113], automated pavement distress detection and classification [114], structural damage detection in buildings [115], and other fields. The development language used in current research is usually Python or another mature language. In contrast, when dealing with large-scale data sets,

the Julia language has inherent advantages in high-performance processing. Therefore, many scholars and engineers use Julia to develop packages for the applications of computer vision. The **Metalhead.jl** [116] package provides computer vision models that run on top of the **Flux** machine learning library. The package **ImageProjectiveGeometry.jl** [117] is intended as a starting point for the development of a library of projective geometry functions for computer vision in Julia. Currently, the package consists of a number of components that could ultimately be separated into individual packages or added to other existing packages.

4.4. Natural language processing (NLP)

NLP employs computational techniques for the purpose of learning, understanding, and producing human language content [118]. It is an important research direction in the field of computer science and artificial intelligence. Modern NLP algorithms are based on machine learning algorithms, especially statistical machine learning algorithms. Many different machine learning algorithms have been applied to NLP tasks, the most representative of which are deep learning algorithms exemplified by CNN [119–122].

Currently, one of the main research tasks of NLP is to investigate the characteristics of human language and establish the cognitive mechanism of understanding and generating language. In addition, new practical applications for processing human language through computer intelligence have been developed. Many researchers and engineers have developed practical application tools or software packages using the Julia language. For example, **LightNLP.jl** [123] is a lightweight NLP toolkit for the Julia language. However, to the best of the authors' knowledge, there are currently no stable library packages developed in the Julia language specifically for NLP.

4.5. Autonomous driving

Machine learning is widely used in autonomous driving, and it mainly focuses on the environmental perception and behavioral decision-making of autonomous vehicles. The application of machine learning in environmental perception belongs to the category of supervised learning. When performing object recognition on images obtained from the surrounding environment of a vehicle, a large number of images with solid objects are required as training data, and then deep learning methods can identify objects from the new images [41,42,102,124]. The application of machine learning in behavioral decision-making generally involves reinforcement learning. Autonomous vehicles need to interact with the environment, and reinforcement learning learns the mapping relationship between the environment and behavior that interacts with the environment from a large amount of sample data. Thus, whenever an autonomous vehicle perceives the environment, it can act intelligently [125,126].

To the best of the authors' knowledge, there are no software packages or solutions specially developed in Julia for autonomous driving. However, the machine learning algorithms used in autonomous driving are currently implemented by researchers in the Julia language. The amount of data obtained by autonomous vehicles is huge, and the processing is complex, but autonomous vehicles have strict requirements for data processing time. High-level languages such as Python and MATLAB are not as efficient in computing as the Julia language, which was specifically developed for high-performance computing. Therefore, we believe that Julia has strong competitiveness as a programming language for autonomous vehicle platforms.

4.6. Graph analytics

Graph analytics is a rapidly developing research field. It combines graph-theoretic, statistics and database technology to model, store, retrieve and analyze graph-structured data. Samsi [127] used subgraph isomorphism to solve the previous scalability difficulties in machine learning, high-performance computing, and visual analysis. The serial implementations of C++, Python, and Pandas and MATLAB are implemented, and their single-thread performance is measured.

LightGraphs.jl is currently the most comprehensive library developed in Julia for graph analysis [128]. **LightGraphs.jl** provides a set of simple, concrete graphical implementations (including undirected and directed graphs) and APIs for developing more complex graphical implementations under the AbstractGraph type.

4.7. Signal processing

The signal processing in communications is the cornerstones of electrical engineering research and other related fields [129, 130]. Python has natural advantages in analyzing complex signal data due to its numerous packages. In addition, the actually collected signals need to be processed before they can be used for analysis. MATLAB provides many signal processing toolboxes, such as spectrum analysis toolbox, waveform viewer, filter design toolbox. Therefore, MATLAB is also a practical tool for signal data processing.

Current and emerging means of communication increasingly rely on the ability to extract patterns from large data sets to support reasoning and decision-making using machine learning algorithms. This calls the use of the Julia language. For example, Srivastava Prakalp et al. [131] designed an end-to-end programmable hybrid signal accelerator, **PROMISE**, for machine learning algorithms. **PROMISE** can use machine learning algorithms described by Julia and generate **PROMISE** code. **PROMISE** can combine multiple signals and accelerate machine learning algorithms.

4.8. Pattern recognition

Pattern recognition is the automatic processing and interpretation of patterns by means of a computer using mathematical technology [132]. With the development of computer technology, it is possible for humans to study the complex process of information-processing, an important form of which is the recognition of the environment and objects by living organisms. The main research directions of pattern recognition are image processing and computer vision, speech information processing, medical diagnosis and biometric authentication technology [133].

Pattern recognition is generally categorized according to the type of learning procedure used to generate the output value. Medical diagnosis is a typical field of pattern recognition applications. Rajsavi et al. [134] used the Julia libraries packages such as **GLM.jl** [135] to predict the mortality rate of diabetic ICU patients through severity indicators. The application case of this pattern recognition was completely written by Julia language. Other typical applications of pattern recognition techniques are automatic speech recognition, text classification , face recognition. **Languages.jl** [136] is a Julia package for working with human languages. Script detection model works by checking the Unicode character ranges present within the input text. But the package was supported only for English and German currently.

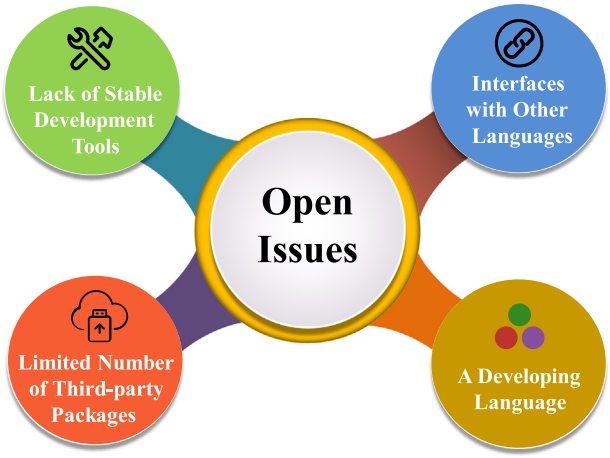

Fig. 7. Open issues of Julia language.

5. Julia in machine learning: Open issues

5.1. Overview

Since its release, the advantages of Julia language, such as simplicity and efficiency, have been recognized by developers in various fields. However, with the promotion of the Julia language and the steady increase in the number of users, it also faces several open issues; see Fig. 7.

5.2. A developing language

Although Julia has developed rapidly, its influence is far less than that of other popular programming languages. After several versions of updates, Julia has become relatively stable, but there are still several problems to be solved. Julia's grammar has changed considerably, and although these changes are for the sake of performance or ease of expression, these differences also make it difficult for different programs to work together. One of Julia's obvious advantages is its satisfactory efficiency; but to write efficient code, one needs to transform the method of thinking in programming and not just copy code into Julia. For people who have just come into contact with Julia, the ease of use can also cause them to ignore this problem and ultimately lead to unsatisfactory code efficiency.

5.3. Lack of stable development tools

Currently, the commonly used editors and IDEs for the Julia language include (1) Juno (Atom Plugin), (2) Visual Studio Code (VS Code Extension), (3) Jupyter (Jupyter kernel), and (4) Jet Brains (IntelliJ IDEA Plugin). According to [137], Juno is currently the most popular editor. These editors and IDEs are extensions based on third-party platforms, which can quickly build development environments for Julia in its early stages of development, but in the long run, this is not a wise approach. Users need to configure Julia initially, but the final experience is not satisfactory. Programming languages such as MATLAB, Python and C/C++ each have their own IDE, which integrates the functions of code writing, analysis, compilation and debugging. Although editors and IDEs have achieved many excellent functions, it is very important to have a complete and Julia-specific IDE.

5.4. Interfacing with other languages

In the process of using Julia for development, although most of the code can be written in Julia, many high-quality, mature numerical computing libraries have been written in C and Fortran. To facilitate the use of existing code, Julia should also make it easy and effective to call C/C++ and Fortran functions. In the field of machine learning, Python has been used to write a large quantity of excellent code. If one desires to transplant code into Julia in a short time for maintenance, simple calls are necessary, which can greatly reduce the learning curve.

Currently, *PyCall* and *Call* are used in Julia to invoke these existing libraries, but Julia still needs a more concise and general invocation method. More important is finding a method to ensure that the parts of these calls can maintain the original execution efficiency or the execution efficiency of the native Julia language. At the same time, it is also very important to embed Julia's code in other languages, which would not only popularize the use of Julia more quickly but also combine the characteristics of Julia to enable researchers to accomplish tasks more quickly.

5.5. Limited number of third-party packages

For good programming languages, the quantity and quality of third-party libraries are very important. For Python, there are 194,934 projects registered in *PyPI* [11], while the number of Julia third-party packages registered in Julia Observer is approximately 3000. The number of third-party libraries in Julia is increasing, but there are still relatively few compared with other programming languages, and there may not be suitable libraries available in some unpopular areas.

Because Julia is still in the early stage of development, version updates are faster, and the interface and grammar of the program are greatly changed in each version upgrade. After the release of Julia 1.0, the Julia language has become more mature and stable than in the past. However, many excellent third-party machine learning libraries were written before the release of Julia 1.0 and failed to update to the new version in time. Users need to carefully evaluate whether a third-party library has been updated to the latest version of Julia to ensure its normal use. In addition, Julia is designed for parallel programming, but there are not many third-party libraries for parallel programming. Currently, the more commonly used third-party packages of Julia are **CUDAnative.jl**, **CuArrays.jl**, and **juliaDB.jl**. However, many functions in these packages are still in the testing stage.

Although Julia libraries are not as rich as those of Python, the prospects for development are optimistic. Officials have provided statistical trends in the number of repositories. Many scholars and technicians are committed to improving the Julia libraries. Rong Hongbo et al. [138] used Julia, Intel MKL and the SPMP library to implement **Sparso**, which is a sparse linear algebra context-driven optimization tool that can accelerate machine learning algorithms. Plumb Gregory et al. [139] compiled a library package for fast Fourier analysis using Julia, which made it easier for fast Fourier analysis to be employed in statistical machine learning algorithms.

6. Conclusions

This paper has systematically investigated the development status of the Julia language in the field of machine learning, including machine learning algorithms written in Julia, the application of the Julia language in machine learning, and the potential challenges faced by Julia. We find that: (1) Machine learning algorithms written in Julia are mainly supervised learning algorithms, and there are fewer algorithms for unsupervised learning. (2) The

Julia language is widely used in seven popular machine learning research topics: pattern recognition, NLP, IoT data analysis, computer vision, autonomous driving, graph analytics, and signal processing. (3) There are far fewer available application packages than there are for other high-level languages, such as Python, which is Julia's greatest challenge. This survey provides a comprehensive investigation of the applications of Julia in machine learning. We believe that with the gradual maturing of the Julia language and the development of related third-party packages, the Julia language will be a highly competitive programming language for machine learning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was jointly supported by the National Natural Science Foundation of China (11602235), the Fundamental Research Funds for China Central Universities (2652018091), and the Major Project for Science and Technology (2020AA002). The authors would like to thank the editor and the reviewers for their valuable comments.

References

[1] M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science 349 (6245) (2015) 255–260, http://dx.doi.org/10.1126/science.aaa8415.
[2] R.C. Deo, Machine learning in medicine, Circulation 132 (20) (2015) 1920–1930, http://dx.doi.org/10.1161/circulationaha.115.001593.
[3] P. Domingos, A few useful things to know about machine learning, Commun. ACM 55 (10) (2012) 78–87, http://dx.doi.org/10.1145/2347736.2347755.
[4] P. Riley, Three pitfalls to avoid in machine learning, Nature 572 (7767) (2019) 27–29, http://dx.doi.org/10.1038/d41586-019-02307-y.
[5] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, http://dx.doi.org/10.1038/nature14539.
[6] E. Mjolsness, D. DeCoste, Machine learning for science: State of the art and future prospects, Science 293 (5537) (2001) 2051–+, http://dx.doi.org/10.1126/science.293.5537.2051.
[7] E. Serrano, J. Garcia Blas, J. Carretero, M. Abella, M. Desco, Medical imaging processing on a big data platform using python: Experiences with heterogeneous and homogeneous architectures, in: 2017 17th Ieee/Acm International Symposium on Cluster, Cloud and Grid Computing, in: IEEE-ACM International Symposium on Cluster Cloud and Grid Computing, 2017, pp. 830–837, http://dx.doi.org/10.1109/ccgrid.2017.56.
[8] Z. Voulgaris, Julia for Data Science, first ed., Technics Publications, LLC, 2016.
[9] O. Dinari, A. Yu, O. Freifeld, J. Fisher, Distributed MCMC inference in Dirichlet process mixture models using Julia, in: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2019, pp. 518–525, http://dx.doi.org/10.1109/CCGRID.2019.00066.
[10] J. Bezanson, A. Edelman, S. Karpinski, V.B. Shah, Julia: A fresh approach to numerical computing, Siam Rev. 59 (1) (2017) 65–98, http://dx.doi.org/10.1137/141000671.
[11] J.M. Perkel, Julia: Come for the syntax, stay for the speed, Nature 572 (7767) (2019) 141–142, http://dx.doi.org/10.1038/d41586-019-02310-3.
[12] [link]. URL https://julialang.org/benchmarks/.
[13] C. Lattner, V. Adve, LLVM: A compilation framework for lifelong program analysis and transformation, 2004, pp. 75–86, http://dx.doi.org/10.1109/cgo.2004.1281665.
[14] Z. Huo, G. Mei, G. Casolla, F. Giampaolo, Designing an efficient parallel spectral clustering algorithm on multi-core processors in Julia, J. Parallel Distrib. Comput. 138 (2020) 211–221, http://dx.doi.org/10.1016/j.jpdc.2020.01.003.
[15] T. Besard, C. Foket, B. De Sutter, Effective extensible programming: unleashing Julia on GPUs, IEEE Trans. Parallel Distrib. Syst. 30 (4) (2018) 827–841, http://dx.doi.org/10.1109/TPDS.2018.2872064.
[16] R. Huang, W. Xu, Y. Wang, S. Liverani, A.E. Stapleton, Performance comparison of Julia distributed implementations of Dirichlet process mixture models, in: Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019, pp. 3350–3354. http://dx.doi.org/10.1109/BigData47090.2019.9005453.
[17] L. Ruthotto, E. Treister, E. Haber, Jlnv-A flexible Julia package for PDE parameter estimation, SIAM J. Sci. Comput. 39 (5) (2017) S702–S722, http://dx.doi.org/10.1137/16m1081063.
[18] W.D. Penny, J. Kilner, F. Blankenburg, Robust Bayesian general linear models, Neuroimage 36 (3) (2007) 661–671, http://dx.doi.org/10.1016/j.neuroimage.2007.01.058.
[19] R. Frigola, Bayesian Time Series Learning with Gaussian Processes, University of Cambridge, 2015, http://dx.doi.org/10.17863/CAM.46480.
[20] C. Strickland, R. Burdett, K. Mengersen, R. Denham, PySSM: A Python module for Bayesian inference of linear Gaussian state space models, J. Stat. Softw. Artic. 57 (6) (2014) 1–37, http://dx.doi.org/10.18637/jss.v057.i06.
[21] U.K. Mertens, A. Voss, S. Radev, ABrox A-user-friendly python module for approximate Bayesian computation with a focus on model comparison, Plos One 13 (3) (2018) http://dx.doi.org/10.1371/journal.pone.0193981.
[22] H. Toosi, A. Moeini, I. Hajirasouliha, BAMSE: Bayesian model selection for tumor phylogeny inference among multiple samples, BMC Bioinformatics 20 (2019) http://dx.doi.org/10.1186/s12859-019-2824-3.
[23] J. Luttinen, BayesPy: Variational Bayesian inference in Python, J. Mach. Learn. Res. 17 (2016) 1–6, https://dl.acm.org/doi/10.5555/2946645.2946686.
[24] A. Patil, D. Huard, C.J. Fonnesbeck, PyMC: Bayesian stochastic modelling in Python, J. Stat. Softw. 35 (4) (2010) 1–81, http://dx.doi.org/10.18637/jss.v035.i04.
[25] V. Jarno, R. Jaakko, H. Jouni, J. Pasi, T. Ville, V. Aki, Bayesian modeling with Gaussian processes using the MATLAB toolbox GPstuff (v3.3), Statistics (2012).
[26] L. Zhang, S. Agravat, G. Derado, S. Chen, B.J. McIntosh, F.D. Bowman, BSMac: A MATLAB toolbox implementing a Bayesian spatial model for brain activation and connectivity, J. Neurosci. Methods 204 (1) (2012) 133–143, http://dx.doi.org/10.1016/j.jneumeth.2011.10.025.
[27] M.M. Cusumano-Towner, V.K.K.V. Mansinghka, A design proposal for Gen: Probabilistic programming with fast custom inference via code generation, in: Proceedings - the 2nd ACM SIGPLAN International Workshop, 2018, p. 57, http://dx.doi.org/10.1145/3211346.3211350.
[28] M. Cox, T. van de Laar, B. de Vries, A factor graph approach to automated design of Bayesian signal processing algorithms, Internat. J. Approx. Reason. 104 (2019) 185–204, http://dx.doi.org/10.1016/j.ijar.2018.11.002.
[29] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D.D. Cox, Hyperopt: A Python library for model selection and hyperparameter optimization, Comput. Sci. Discovery 8 (2015) 1, http://dx.doi.org/10.1088/1749-4699/8/1/014008.
[30] [link]. URL https://github.com/KristofferC/NearestNeighbors.jl.
[31] L. Breiman, Random forests machine learning, Mach. Learn. 45 (2001) 5–32, http://dx.doi.org/10.1023/A:1010933404324.
[32] T.K. Ho, Random decision forests, in: Proceedings of 3rd International Conference on Document Analysis and Recognition, Vol. 1, 1995, pp. 278–282, http://dx.doi.org/10.1109/ICDAR.1995.598994.
[33] Y. Zhou, P. Gallins, A review and tutorial of machine learning methods for microbiome host trait prediction, Front. Genet. 10 (2019) 579, http://dx.doi.org/10.3389/fgene.2019.00579.
[34] A. Upadhyay, A. Shetty, S. Kumar Singh, Z. Siddiqui, Land use and land cover classification of LISS-III satellite image using KNN and decision tree, in: 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 1277–1280, https://ieeexplore.ieee.org/document/7724471.
[35] T. Keck, FastBDT: A speed-optimized and cache-friendly implementation of stochastic gradient-boosted decision trees for multivariate classification, 2016, arXiv:1609.06119.
[36] F. Yang, X. Han, J. Lang, W. Lu, L. Liu, L. Zhang, J. Pan, Acm, Commodity Recommendation for Users Based on E-commerce Data, in: Proceedings of the 2018 2nd International Conference on Big Data Research, 2018, pp. 146–149, http://dx.doi.org/10.1145/3291801.3291803.
[37] I. Seiferling, N. Naik, C. Ratti, R. Proulx, Green streets - Quantifying and mapping urban trees with street-level imagery and computer vision, Landsc. Urban Plan. 165 (2017) 93–101, http://dx.doi.org/10.1016/j.landurbplan.2017.05.010.
[38] V. Vapnik, The Nature of Statistical Learning Theory, Springer science and business media, 2013.
[39] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: primal estimated sub-gradient solver for SVM, Math. Program. 127 (1) (2011) 3–30, http://dx.doi.org/10.1007/s10107-010-0420-4.
[40] C. Chang, C. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 27, http://dx.doi.org/10.1145/1961189.1961199.

[41] P.M. Kebria, A. Khosravi, S.M. Salaken, S. Nahavandi, Deep imitation learning for autonomous vehicles based on convolutional neural networks, IEEE-CAA J. Autom. Sin. 7 (1) (2020) 82–95, http://dx.doi.org/10.1109/jas.2019.1911825.
[42] Y. Parmar, S. Natarajan, G. Sobha, DeepRange: deep-learning-based object detection and ranging in autonomous driving, IET Intell. Transp. Syst. 13 (8) (2019) 1256–1264, http://dx.doi.org/10.1049/iet-its.2018.5144.
[43] [link]. URL https://github.com/mpastell/LIBSVM.jl.
[44] J. Gwak, J. Jung, R. Oh, M. Park, M.A.K. Rakhimov, J. Ahn, A review of intelligent self-driving vehicle software research, Ksii Trans. Internet Inf. Syst. 13 (11) (2019) 5299–5320, http://dx.doi.org/10.3837/tiis.2019.11.002.
[45] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, G. Varoquaux, Machine learning for neuroirnaging with scikit-learn, Front. Neuroinform. 8 (2014) 14, http://dx.doi.org/10.3389/fninf.2014.00014.
[46] A. Jovic, K. Brkic, N. Bogunovic, An overview of free software tools for general data mining, in: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014, pp. 1112–1117, http://dx.doi.org/10.1109/MIPRO.2014.6859735.
[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[48] J. Demsar, T. Curk, A. Erjavec, C. Gorup, T. Hocevar, M. Milutinovic, M. Mozina, M. Polajnar, M. Toplak, A. Staric, M. Stajdohar, L. Umek, L. Zagar, J. Zbontar, M. Zitnik, B. Zupan, Orange: Data mining toolbox in Python, J. Mach. Learn. Res. 14 (2013) 2349–2353.
[49] J. Demsar, B. Zupan, G. Leban, T. Curk, Orange: From experimental machine learning to interactive data mining, in: J.F. Boulicaut, F. Esposito, F. Giannotti, D. Pedreschi (Eds.), Knowledge Discovery in Databases: Pkdd 2004, Proceedings, in: Lecture Notes in Artificial Intelligence, vol. 3202, 2004, pp. 537–539.
[50] Y.H. Shan, W.F. Lu, C.M. Chew, Pixel and feature level based domain adaptation for object detection in autonomous driving, Neurocomputing 367 (2019) 31–38, http://dx.doi.org/10.1016/j.neucom.2019.08.022.
[51] E. Arnold, O.Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, A. Mouzakitis, A survey on 3D object detection methods for autonomous driving applications, IEEE Trans. Intell. Transp. Syst. 20 (10) (2019) 3782–3795, http://dx.doi.org/10.1109/tits.2019.2892405.
[52] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[53] Y. Raja, S. McKenna, S. Gong, Segmentation and tracking using colour mixture models, Lecture Notes in Comput. Sci. 1351 (1998) 607–614, http://dx.doi.org/10.1007/3-540-63930-6_173.
[54] M. Bruneau, T. Mottet, S. Moulin, M. Kerbiriou, F. Chouly, S. Chretien, C. Guyeux, A clustering package for nucleotide sequences using Laplacian Eigenmaps and Gaussian mixture model, Comput. Biol. Med. 93 (2017) 66–74, http://dx.doi.org/10.1016/j.compbiomed.2017.12.003.
[55] T.W.S. Holoien, P.J. Marshall, R.H. Wechsler, EmpiriciSN: Re-sampling observed supernova/host galaxy populations using an XD Gaussian mixture model, Astron. J. 153 (6) (2017) http://dx.doi.org/10.3847/1538-3881/aa68a1.
[56] [link]. URL https://github.com/AmebaBrain/GmmFlow.jl.
[57] [link]. URL https://github.com/davidavdav/GaussianMixtures.jl.
[58] F. Srajer, Z. Kukelova, A. Fitzgibbon, A benchmark of selected algorithmic differentiation tools on some problems in computer vision and machine learning, Optim. Methods Softw. 33 (4–6) (2018) 889–906, http://dx.doi.org/10.1080/10556788.2018.1435651.
[59] G. Douzas, F. Bacao, F. Last, Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE, Inform. Sci. 465 (2018) 1–20, http://dx.doi.org/10.1016/j.ins.2018.06.056.
[60] S. Yu, L.C. Tranchevent, X.H. Liu, W. Glanzel, J.A.K. Suykens, B. De Moor, Y. Moreau, Optimized data fusion for kernel k-means clustering, IEEE Trans. Pattern Anal. Mach. Intell. 34 (5) (2012) 1031–1039, http://dx.doi.org/10.1109/tpami.2011.255.
[61] K. Zhang, Y.X. Zhu, S.P. Leng, Y.J. He, S. Maharjan, Y. Zhang, Deep learning empowered task offloading for mobile edge computing in urban informatics, IEEE Internet Things J. 6 (5) (2019) 7635–7647, http://dx.doi.org/10.1109/jiot.2019.2903191.
[62] F. Corpet, Multiple sequence alignment with hierarchical-clustering, Nucleic Acids Res. 16 (22) (1988) 10881–10890, http://dx.doi.org/10.1093/nar/16.22.10881.
[63] S.C. Johnson, Hierarchical clustering schemes, Psychometrika 32 (3) (1967) 241–254, http://dx.doi.org/10.1007/bf02289588.
[64] G. Karypis, E.H. Han, V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer 32 (8) (1999) 68–+, http://dx.doi.org/10.1109/2.781637.
[65] D. Jaeger, J. Barth, A. Niehues, C. Fufezan, PyGCluster, a novel hierarchical clustering approach, Bioinformatics 30 (6) (2014) 896–898, http://dx.doi.org/10.1093/bioinformatics/btt626.
[66] D. Muellner, Fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python, J. Stat. Softw. 53 (9) (2013) 1–18, http://dx.doi.org/10.18637/jss.v053.i09.
[67] J. Kabzan, L. Hewing, A. Liniger, M.N. Zeilinger, Learning-based model predictive control for autonomous racing, Ieee Robot. Autom. Lett. 4 (4) (2019) 3363–3370, http://dx.doi.org/10.1109/lra.2019.2926677.
[68] S. Datta, Hierarchical stellar clusters in molecular clouds, in: R. DeGrijs, J.R.D. Lepine (Eds.), Star Clusters: Basic Galactic Building Blocks Throughout Time and Space, in: IAU Symposium Proceedings Series, 2010, pp. 377–379, http://dx.doi.org/10.1017/s1743921309991396.
[69] G. Kerr, H.J. Ruskin, M. Crane, P. Doolan, Techniques for clustering gene expression data, Comput. Biol. Med. 38 (3) (2008) 283–293, http://dx.doi.org/10.1016/j.compbiomed.2007.11.001.
[70] J. Jacques, C. Biernacki, Model-based co-clustering for ordinal data, Comput. Statist. Data Anal. 123 (2018) 101–115, http://dx.doi.org/10.1016/j.csda.2018.01.014.
[71] A. Pessia, J. Corander, Kpax3: Bayesian bi-clustering of large sequence datasets, Bioinformatics 34 (12) (2018) 2132–2133, http://dx.doi.org/10.1093/bioinformatics/bty056.
[72] [link]. URL https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.
[73] [link]. URL https://www.alglib.net/dataanalysis/principalcomponentsanalysis.php.
[74] [link]. URL https://docs.opencv.org/master/d1/dee/tutorial_introduction_to_pca.html.
[75] [link]. URL https://github.com/JuliaStats/MultivariateStats.jl.
[76] S. Vorobyov, A. Cichocki, Blind noise reduction for multisensory signals using ICA and subspace filtering, with application to EEG analysis, Biol. Cybernet. 86 (4) (2002) 293–303, http://dx.doi.org/10.1007/s00422-001-0298-6.
[77] X. Ren, X. Hu, Z. Wang, Z. Yan, MUAP extraction and classification based on wavelet transform and ICA for EMG decomposition, Med. Biol. Eng. Comput. 44 (5) (2006) 371–382, http://dx.doi.org/10.1007/s11517-006-0051-3.
[78] E.B. Assi, S. Rihana, M. Sawan, Kmeans-ICA based automatic method for ocular artifacts removal in a motorimagery classification, in: 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2014, pp. 6655–6658, http://dx.doi.org/10.1109/EMBC.2014.6945154.
[79] [link]. URL https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.htm.
[80] [link]. URL https://www.mathworks.com/matlabcentral/fileexchange/38300-pca-and-ica-package?s_tid=prof_contriblnk.
[81] M. Fey, J.E. Lenssen, Fast graph representation learning with PyTorch geometric, 2019, arXiv:1903.02428.
[82] J. Shen, P. Nguyen, Y. Wu, Z. Chen, M.X. Chen..., Lingvo: a modular and scalable framework for sequence-to-sequence modeling, 2019, arXiv:1902.08295.
[83] Y.-P. Tang, G.-X. Li, S.-J. Huang, ALiPy: Active learning in Python, 2019, arXiv:1901.03802.
[84] J. Tang, L. Ericson, J. Folkesson, P. Jensfelt, GCNv2: Efficient correspondence prediction for real-time SLAM, IEEE Robot. Autom. Lett. 4 (4) (2019) 3505–3512, http://dx.doi.org/10.1109/LRA.2019.2927954.
[85] Z. Huang, L. Huang, Y. Gong, C. Huang, X. Wang, Mask scoring R-CNN, 2019, arXiv:1903.00241.
[86] N. Frazier-Logue, S.J. Hanson, Dropout is a special case of the stochastic delta rule: faster and more accurate deep learning, 2018, arXiv:1808.03578.
[87] S.J. Hanson, A stochastic version of the delta rule, Phys. D 42 (1990) 265–272, http://dx.doi.org/10.1016/0167-2789(90)90081-Y.
[88] L. Luo, Y. Xiong, Y. Liu, X. Sun, Adaptive gradient methods with dynamic bound of learning rate, 2019, arXiv:1902.09843.
[89] M.D. Bloice, C. Stocker, A. Holzinger, Augmentor: An image augmentation library for machine learning, 2017, arXiv:1708.04680.
[90] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, Commun. Acm 60 (6) (2017) 84–90, http://dx.doi.org/10.1145/3065386.
[91] Y.-R. Liu, Y.-Q. Hu, H. Qian, Y. Yu, C. Qian, ZOOpt: Toolbox for derivative-free optimization, 2017, arXiv:1801.00329.
[92] [link]. URL https://github.com/denizyuret/Knet.jl.
[93] [link]. URL https://github.com/FluxML/Flux.jl.
[94] [link]. URL https://github.com/malmaud/TensorFlow.jl.
[95] G. Ditzler, J.C. Morrison, Y. Lan, G.L. Rosen, Fizzy: Feature subset selection for metagenomics, 16 (2015) 358. http://dx.doi.org/10.1186/s12859-015-0793-8.

[96] B. Wicht, A. Fischer, J. Hennebert, DLL: A fast deep neural network library, in: L. Pancioni, F. Schwenker, E. Trentin (Eds.), Artificial Neural Networks in Pattern Recognition, Annpr 2018, in: Lecture Notes in Artificial Intelligence, vol. 11081, 2018, pp. 54–65, http://dx.doi.org/10.1007/978-3-319-99978-4_4.
[97] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, V. Dixit, DiffEqFlux.jl - A Julia library for neural differential equations, 2019, arXiv:1902.02376.
[98] [link]. URL https://github.com/JuliaDiffEq/DifferentialEquations.jl.
[99] G.B. Huang, Q.Y. Zhu, C.K. Siew, Extreme learning machine: A new learning scheme of feedforward neural networks, in: 2004 IEEE International Joint Conference on Neural Networks, Vols 1-4, Proceedings, in: IEEE International Joint Conference on Neural Networks (IJCNN), 2004, pp. 985–990, http://dx.doi.org/10.1109/IJCNN.2004.1380068.
[100] L.L.C. Kasun, H. Zhou, G.-B. Huang, C.M. Vong, Representational learning with ELMs for big data, IEEE Intell. Syst. 28 (6) (2013) 31–34.
[101] J. Tang, C. Deng, G.-B. Huang, Extreme learning machine for multilayer perceptron, IEEE Trans. Neural Netw. Learn. Syst. 27 (4) (2015) 809–821, http://dx.doi.org/10.1109/TNNLS.2015.2424995.
[102] Z.C. Ouyang, J.W. Niu, Y. Liu, M. Guizani, Deep CNN-based real-time traffic light detector for self-driving vehicles, IEEE Trans. Mob. Comput. 19 (2) (2020) 300–313, http://dx.doi.org/10.1109/tmc.2019.2892451.
[103] I. Lee, K. Lee, The Internet of Things (IoT): Applications, investments, and challenges for enterprises, Bus. Horiz. 58 (4) (2015) 431–440, http://dx.doi.org/10.1016/j.bushor.2015.03.008.
[104] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst.-Int. J. Esci. 29 (7) (2013) 1645–1660, http://dx.doi.org/10.1016/j.future.2013.01.010.
[105] L. Atzori, A. Iera, G. Morabito, The Internet of Things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805, http://dx.doi.org/10.1016/j.comnet.2010.05.010.
[106] G. Mei, N. Xu, J. Qin, B. Wang, P. Qi, A survey of Internet of Things (IoT) for geo-hazards prevention: Applications, technologies, and challenges, IEEE Internet Things J. (2019) 1, http://dx.doi.org/10.1109/JIOT.2019.2952593.
[107] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: A survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960, http://dx.doi.org/10.1109/comst.2018.2844341.
[108] M.S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, A.P. Sheth, Machine learning for Internet of Things data analysis: A survey, Digit. Commun. Netw. 4 (3) (2018) 161–175, http://dx.doi.org/10.1016/j.dcan.2017.10.002.
[109] [link]. URL https://invenia.github.io/blog/.
[110] [link]. URL https://juliacomputing.com/case-studies/fugro-roames-ml.html.
[111] B.T. Grys, D.S. Lo, N. Sahin, O.Z. Kraus, Q. Morris, C. Boone, B.J. Andrews, Machine learning and computer vision approaches for phenotypic profiling, J. Cell Biol. 216 (1) (2017) 65–71, http://dx.doi.org/10.1083/jcb.201610026.
[112] N. Naik, S.D. Kominers, R. Raskar, E.L. Glaeser, C.A. Hidalgo, Computer vision uncovers predictors of physical urban change, Proc. Natl. Acad. Sci. USA 114 (29) (2017) 7571–7576, http://dx.doi.org/10.1073/pnas.1619003114.
[113] A.K. Patel, S. Chatterjee, Computer vision-based limestone rock-type classification using probabilistic neural network, Geosci. Front. 7 (1) (2016) 53–60, http://dx.doi.org/10.1016/j.gsf.2014.10.005.
[114] K. Gopalakrishnan, S.K. Khaitan, A. Choudhary, A. Agrawal, Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection, Constr. Build. Mater. 157 (2017) 322–330, http://dx.doi.org/10.1016/j.conbuildmat.2017.09.110.
[115] Y.J. Cha, J.G. Chen, O. Buyukozturk, Output-only computer vision based damage detection using phase-based optical flow and unscented kalman filters, Eng. Struct. 132 (2017) 300–313, http://dx.doi.org/10.1016/j.engstruct.2016.11.038.
[116] [link]. URL https://github.com/FluxML/Metalhead.jl.
[117] [link]. URL https://github.com/peterkovesi/ImageProjectiveGeometry.jl.
[118] J. Hirschberg, C.D. Manning, Advances in natural language processing, Science 349 (6245) (2015) 261–266, http://dx.doi.org/10.1126/science.aaa8685.
[119] S. Poria, E. Cambria, A. Gelbukh, Aspect extraction for opinion mining with a deep convolutional neural network, Knowl.-Based Syst. 108 (2016) 42–49, http://dx.doi.org/10.1016/j.knosys.2016.06.009.
[120] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75, http://dx.doi.org/10.1109/mci.2018.2840738.
[121] M. Gimenez, J. Palanca, V. Botti, Semantic-based padding in convolutional neural networks for improving the performance in natural language processing. A case of study in sentiment analysis, Neurocomputing 378 (2020) 315–323, http://dx.doi.org/10.1016/j.neucom.2019.08.096.
[122] W.B. Liu, Z.D. Wang, X.H. Liu, N.Y. Zengb, Y.R. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26, http://dx.doi.org/10.1016/j.neucom.2016.12.038.
[123] [link]. URL https://github.com/hshindo/LightNLP.jl.
[124] H.F. Liu, X.F. Han, X.R. Li, Y.Z. Yao, P. Huang, Z.M. Tang, Deep representation learning for road detection using Siamese network, Multimedia Tools Appl. 78 (17) (2019) 24269–24283, http://dx.doi.org/10.1007/s11042-018-6986-1.
[125] L.G. Cuenca, E. Puertas, J.F. Andres, N. Aliane, Autonomous driving in roundabout maneuvers using reinforcement learning with Q-learning, Electronics 8 (12) (2019) 13, http://dx.doi.org/10.3390/electronics8121536.
[126] C. Desjardins, B. Chaib-draa, Cooperative adaptive cruise control: A reinforcement learning approach, IEEE Trans. Intell. Transp. Syst. 12 (4) (2011) 1248–1260, http://dx.doi.org/10.1109/tits.2011.2157145.
[127] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, D. Staheli, J. Kepner, Static graph challenge: Subgraph isomorphism, in: 2017 IEEE High Performance Extreme Computing Conference (HPEC), 2017, pp. 1–6, http://dx.doi.org/10.1109/HPEC.2017.8091039.
[128] [link]. URL https://github.com/JuliaGraphs/LightGraphs.jl.
[129] B. Uengtrakul, D. Bunnjaweht, Ieee, A cost efficient software defined radio receiver for demonstrating concepts in communication and signal processing using Python and RTL-SDR, in: 2014 Fourth International Conference on Digital Information and Communication Technology and It's Applications, in: International Conference on Digital Information and Communication Technology and it's Applications, 2014, pp. 394–399.
[130] K. Gideon, C. Nyirenda, C. Temaneh-Nyah, Echo state network-based radio signal strength prediction for wireless communication in Northern Namibia, Iet Commun. 11 (12) (2017) 1920–1926, http://dx.doi.org/10.1049/iet-com.2016.1290.
[131] P. Srivastava, M. Kang, S.K. Gonugondla, S. Lim, J. Choi, V. Adve, N.S. Kim, N. Shanbhag, PROMISE: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms, in: 2018 Acm/Ieee 45th Annual International Symposium on Computer Architecture, in: Conference Proceedings Annual International Symposium on Computer Architecture, 2018, pp. 43–56, http://dx.doi.org/10.1109/isca.2018.00015.
[132] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[133] R. Milewski, V. Govindaraju, Binarization and cleanup of handwritten text from carbon copy medical form images, Pattern Recognit. 41 (4) (2008) 1308–1315, http://dx.doi.org/10.1016/j.patcog.2007.08.018.
[134] R.S. Anand, P. Stey, S. Jain, D.R. Biron, H. Bhatt, K. Monteiro, E. Feller, M.L. Ranney, I.N. Sarkar, E.S. Chen, Predicting mortality in diabetic ICU patients using machine learning and severity indices, AMIA Jt. Summits Transl. Sci. Proc. 2017 (2018) 310–319.
[135] [link]. URL https://github.com/JuliaStats/GLM.jl.
[136] [link]. URL https://github.com/JuliaText/Languages.jl.
[137] [link]. URL https://julialang.org/blog/2019/08/2019-julia-survey.
[138] H. Rong, J. Park, L. Xiang, T.A. Anderson, M. Smelyanskiy, Sparso: Context-driven Optimizations of Sparse Linear Algebra, in: 2016 International Conference on Parallel Architecture and Compilation Techniques, 2016, pp. 247–259, http://dx.doi.org/10.1145/2967938.2967943.
[139] G. Plumb, D. Pachauri, R. Kondor, V. Singh, SnFFT: A Julia toolkit for Fourier analysis of functions over permutations, J. Mach. Learn. Res. 16 (2015) 3469–3473.