Electronics and Electrical Engineering

AIMS Electronics and Electrical Engineering, 5(1): 68–92.
DOI: 10.3934/electreng.2021005
Received: 03 November 2020
Accepted: 11 January 2021
Published: 03 February 2021

http://www.aimspress.com/journal/ElectrEng

Review

Proposed big data architecture for facial recognition using machine learning

Suriya Priya R Asaithambi[1], Sitalakshmi Venkatraman[2,*] and Ramanathan Venkatraman[1]

[1] Institute of Systems Science, National University of Singapore, Singapore
[2] Department of Information Technology, Melbourne Polytechnic, VIC, Australia

*** Correspondence:** Email: sitavenkat@melbournepolytechnic.edu.au; Tel: +61892663143.

Abstract: With the abundance of raw data generated from various sources including social networks, big data has become essential in acquiring, processing, and analyzing heterogeneous data from multiple sources for real-time applications. In this paper, we propose a big data framework suitable for pre-processing and classification of image as well as text analytics by employing two key workflows, called big data (BD) pipeline and machine learning (ML) pipeline. Our unique end-to-end workflow integrates data cleansing, data integration, data transformation and data reduction processes, followed by various analytics using suitable machine learning techniques. Further, our model is the first of its kind to augment facial recognition with sentiment analysis in a distributed big data framework. The implementation of our model uses state-of-the-art distributed technologies to ingest, prepare, process and analyze big data for generating actionable data insights by employing relevant ML algorithms such as k-NN, logistic regression and decision tree. In addition, we demonstrate the application of our big data framework to facial recognition system using open sources by developing a prototype as a use case. We also employ sentiment analysis on non-repetitive semi structured public data (text) such as user comments, image tagging, and other information associated with the facial images. We believe our work provides a novel approach to intersect Big Data, ML and Face Recognition and would create new research to alleviate some of the challenges associated with big data processing in real world applications.

Keywords: big data; machine learning; social networks; sentiment analysis; facial recognition; distributed computing

1. Introduction

In the age of innovation and digital transformation, data is generated in huge volumes and in an increasing velocity that constitute a recently popular term, 'big data'. Recently, big data (BD) related technologies have developed into a hotspot that attracts great attention from academia, industry and even governments around the world. However, three of the key features of big data (3V's), namely multi-sources (Variety), huge volume (Volume) and fast-changing (Velocity), make it difficult for traditional data processing methods such as data mining to effectively support the processing of heterogeneous big data. To address the computational complexity of big data applications, there is a need to explore new approaches for building scalable big data processing architecture. Apache Spark along with tools from Hadoop eco system, enables complex analytics processing using in-memory computational techniques. The principal advantage of such *big data* technologies [1–3] is in their ability to provide computation-intensive operations upon massive data sets in real-time with significant accuracy and performance. Therefore, big data technologies could be considered ideal for facial recognition applications that warrant resource-intensive image analysis using machine learning (ML) algorithms on large corpus of image data collected from multiple internal and external big data sources. However, big data [4,5] and facial recognition [6] form two disparate advancements in technologies that are coming into some common convergence only recently.

Facial recognition with machine learning capabilities is a hot research topic due to its various applications in social media, surveillance systems, online shopping, banking, law enforcement, personalized marketing and access control for Internet of Things within various radical real world scenarios For instance, recent popularity of social networking hubs are requiring security and law enforcement in future to apply Artificial Intelligence (AI) on big data streams of facial data in real-time. AI-enabled facial recognition is required by retail and banking industries to understand consumer behavior patterns to improve their personalized products and services. While there is a plethora of facial recognition technologies, they need to be adapted with the massive upsurge of social networks and Internet of Things that are accompanied by big data of facial images stored and retrieved from several intertwined and disparate real-world application domains. Further, the process of facial recognition from a large set of images or videos is complex. Classical ML approaches involve domain knowledge of the data to create features, and such techniques are not applicable for the radical applications of the future. There are practical challenges of real-time processes apart from interpersonal variations due to similarities between two persons such as twins, or intrapersonal variations due to differences in two different image data of the same person contributed by several factors such as pose, obstruction, age, expression, quality and noise. Modern ML approaches require automatic feature extraction from large image data sets that remain invariant to such variations by adopting novel deep learning techniques. This forms the main motivation of this research work to propose a big data framework for providing an effective solution to the said problem.

In this paper, we leverage on recent developments in public large datasets, social networking public media and relevant ML algorithms to transform the conventional view of addressing facial recognition issues with a contemporary perspective. Our key contributions of this work are three-fold forming a modest initial step towards advancing an important research in this direction. These are given below

1. An approach first of its kind to intersect big data, machine language and facial recognition through the proposal of a big data architecture by employing two main workflow processes, namely BD pipeline and ML pipeline.

2. Application of the proposed BD architecture for developing a novel facial recognition prototype as a use case. We develop a prototype as an amalgamation of BD pipeline to include image data pre-processing, real time data streaming along with ML pipeline processing on stored facial images against real time streams.

3. Our unique method augments image analysis with text analytics on selected attributes such as social media tweet, and face tagging associated with social networks towards improving facial recognition.

The rest of the paper is organized as follows. In Section 2 we provide a review of related work and the unique contribution of our work. Section 3 describes our proposed big data architecture with the details of the solution model using relevant big data technologies. In Section 4, we demonstrate the application of proposed model for an effective facial recognition solution using machine learning. Finally, we provide the conclusion of our study in Section 5.

2. Related work

The global market for software taking benefit of facial recognition is expected to grow from $3.85 billion USD in 2017 to $9.78 billion USD by 2023. The Asia Pacific region, which holds around 16% of its market share, is the fastest-growing region [7]. This section is divided into three parts. In these parts, we provide an overview of some exiting works done from three perspectives: i) feature engineering and hyper parameter tuning for image or video processing, ii) facial recognition and learning algorithms, and iii) big data architecture and technologies.

2.1. Feature engineering and hyper parameter tuning for images and videos

Feature selection reduces the dataset by removing irrelevant or redundant features. Based on the feature extraction techniques used, face recognition uses global or local feature extraction methods. Global feature extraction algorithms used by researchers include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Canonical Correlation Analysis (CCA), and Two Dimensional PCA (2DPCA) [8]. Local feature extraction includes PCA, Support Vector Machines (SVM), Local Binary Pattern (LBP), and Local Binary Pattern Histogram Wavelet Feature (LBPWT) [9]. Global features can recognize pervasive features in the image or video such as texture, shape and other background information. Local features can use them for guidance and focus on smaller subset of processing.

Recently, a k-NN algorithmic variant was employed for expression mining using facial image tagging and classification in the Hadoop and MapReduce environment as a cloud hosting [10]. The experiment used 3120 images of 120 persons (65 male and 55 female candidates) from the AR public Face database using PCA, CCA and LDA combination. This is quite contrary to other studies conducted historically where PCA has been considered as the performant popular choice for facial image processing [11,12].

The combination of multiple local features can also improve the accuracy of face recognition. However, one shortcoming observed is that the local features tend to be sensitive, which makes them vulnerable to local lighting, expression, posture and other factors, and lack of robustness. Therefore, the best feature engineering models that deal with facial recognition effectively try to combine the advantages of both local features and global features. Our work attempts to combine selected global and local feature algorithms as appropriate.

2.2. Facial recognition using learning algorithms

Facial recognition is a technology for identifying or verifying a person in images or videos. The first face recognition [13] algorithm published in 1991 used eigenfaces [12]. In the past, Parkhi et al. used a convolutional neural network (CNN) [14], and He et al. used Laplacian faces [15]. However, in recent works, very Deep Neural Network [16] is observed to be in use towards achieving lightweight performance and high accuracy rates from loss correction and hyper parameter tuning [17–22].

Generally, the process of facial recognition is performed in two key steps: (1) Feature extraction and selection, and (2) Classification of objects. Recent deep learning developments have introduced several other methods, such as the use of facial recognition algorithms, three-dimensional recognition, skin texture analysis, and thermal cameras. While deep learning for facial recognition is well researched in the field, its influence on feature engineering and hyper parameter tuning lack the required importance and exploration. This paper attempts to fill that gap by combining deep learning algorithms with enhanced feature tuning.

2.3. Big data architecture and technologies

Big data architecture and technologies are being well researched in certain domains such as HealthCare [23], Smart Initiatives [24–26], and Climate Change [27]. Researchers have published technical stacks as big data architecture templates using combination of tools from the Hadoop Eco System and distributed computing frameworks such as Spark, Flink and Beam. However, there are a few research observations regarding the hybrid architectural models that use batch as well as real-time processing towards facial recognition in the literature. Our work is a humble step towards bridging this gap.

3. Proposed big data architecture

Training complex face recognition on images and videos can take hours, days, or even weeks. In most cases, a single multi-GPU machine is enough to train large models in a reasonable amount of time. However, for more demanding real-time face recognition workloads, spreading computational loads across multiple machines can dramatically reduce training time, enabling rapid iterative experimentation, and accelerating deep learning deployments. Thus, big data processing architectures and parallel processing frameworks like MapReduce and Spark play a key role in such frontiers.

3.1. Big data technologies

There are two main approaches to enhance facial recognition, namely model parallelism and data parallelism. The big data architecture relies on distributed cluster computing for pre-processing and classification tasks of the heterogeneous and disparate big data streams emanating from various data sources. In this context, data parallelism can be achieved from the regular Hadoop MapReduce stack as proposed in our technical stack. However, when it comes to model parallelism, in memory frameworks play a key role. Thus, we propose to use the Spark in-memory processing to achieve learning model parallelism.

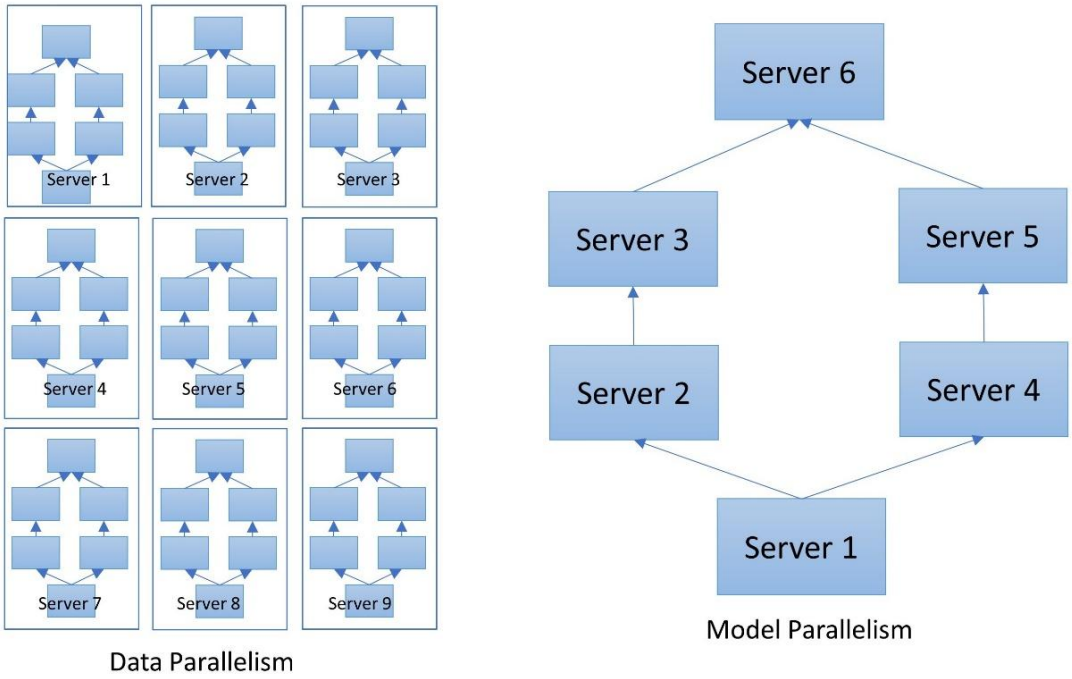

Figure 1. Data vs model parallelism.

To support both data and model parallelism, we adopt the common open-source choice of technologies, namely Hadoop distributed file system and HBase for collecting data from heterogeneous data sources in our proposed big data architecture. Our approach is to use a hybrid of both data parallelism and model parallelism illustrated in Figure 1, where the parallelism can be achieved using MapReduce framework of Hadoop. MapReduce is a programming model for processing large datasets that are employed in a variety of real-world tasks. Developers can specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks [10,28,29]. However, research in the past have only focused on the feature-based batch implementation of facial recognition using MapReduce [29–31]. However, for performance reasons we have chosen Spark over MapReduce for the facial recognition algorithms implemented in this work.

Apache Spark [32] achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine. The research carried out by Hazarika et al. [33] indicates that Spark can solve learning problems on large image data sets using in-memory iterative caching ML libraries that run on top of Spark Data Frames. In some cases, Spark outperforms Hadoop MapReduce in computational speed by 10-times in iterative ML tasks and up to 20-times for iterative applications [34]. This is because Spark leverages in-memory processing as compared to MapReduce which must read from and write to the disk.

3.2. Proposed solution model

The big data processing system we propose is powerful enough to identify or verify a face or understand a facial expression from digital images and videos, which we generally term as facial

recognition. This system works by comparing the most common and prominent facial features from a given image (collected in real-time from various external sources) with the faces stored in a database. The facial recognition system also has the ability to understand patterns and variations based on an individual's facial textures and shape to uniquely recognize a person.

We provide an architectural diagram of our proposed big data processing solution in Figure 2. It shows an end-to-end prototype architecture to ingest, cleanse, process and visualize facial recognition results for generating actionable insights (image is considered as an unstructured big data in our use case).

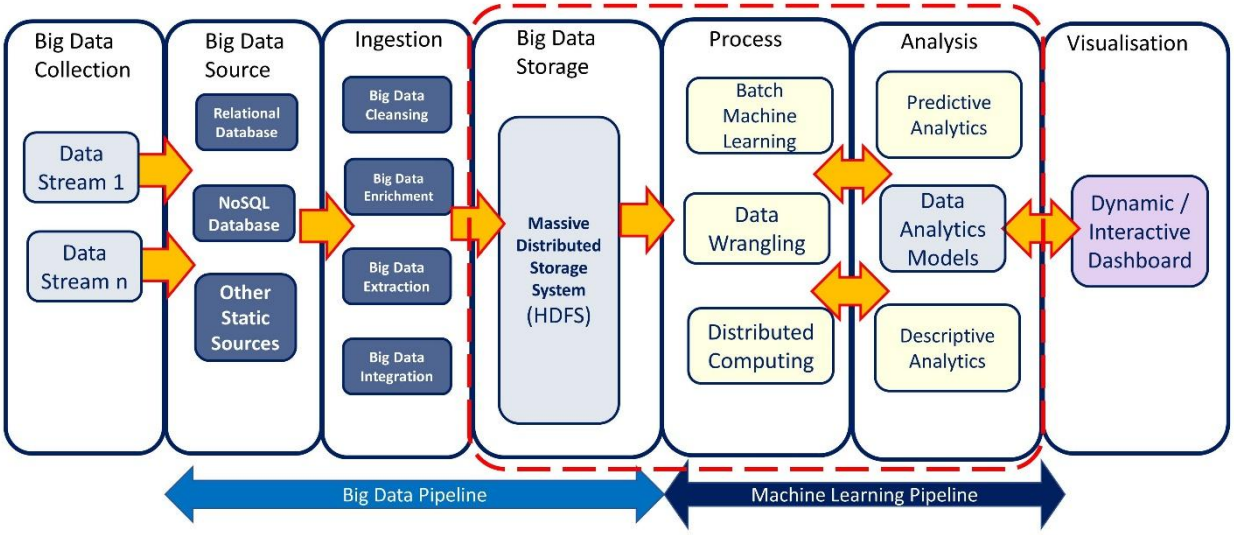

Figure 2. Proposed big data architecture.

Our proposed architecture integrates and automates ingestion, storage, processing and special analysis layers. The prototype consists of two main workflows to effectively process different data streams, namely (1) Big data pipeline and (2) Machine learning pipeline. We describe these two workflows below.

1) Big Data Pipeline

The main role of the big data (BD) pipeline is to automate the movement of huge data of different data types from external sources of data streams into a data lake for downstream analysis. In addition to moving data, we also use data from traditional relational databases, NoSQL databases and static data stores such as jpg/ mp3/ mp4/ json files so that these can be analysed by downstream processes more efficiently in the data lake. The key considerations of tools and techniques to implement the BD pipeline for our proposed big data architecture is that it can meet the following objectives:

- to connect multiple types of data and file stores (e.g., relational & NoSQL databases, static file store for unstructured and semi-structured data), and
- to provide custom scripting support for implementing data quality checks and pre-processing during the data ingestion phase.

2) Machine Learning Pipeline

The main role of the machine learning (ML) pipeline is to help automate the machine learning and parameter tuning of workflows as ML refers to learning patterns in the data. In other words, ML

can infer the pattern or nontrivial relationships between a set of observations and a desired response. The key considerations of tools and techniques to implement the ML pipeline for our proposed architecture is that it can meet the following objectives:

- to support distributed computing so that a scalable pipeline is achieved, and
- to provide functions and API's to implement feature analysis and iterative ML algorithms.

Using the abovementioned pipelines, our proposed big data architecture is generic enough to make use of different sources of big data related to any application requirement including web crawlers to scrap information from real-time data streams / online public datasets as part of the data acquisition process. Lastly, we have also explored the management perspective of the entire solution by considering open-source workflow tools such as Apache NiFi and multi-node cluster management stack from Cloudera under academic license partnership.

3.3. Big data pre-processing

Data pre-processing is a quintessential step in data mining as it affects the quality of insights derived from the ML algorithms applied. A good face preprocessing stage will help improve the reliability of the whole face recognition system. In our proposed architecture, the big data pre-processing consists of the following five steps:

i. Data Imputation: Data imputation involves representing missing values in a dataset. Imputation replaces the missing values with an estimate, and then analyses the full data set as if the imputed values were actual observed values.

ii. Feature Selection: Automatic selection of attributes in data set that would help in pattern recognition are more relevant to face recognition systems.

iii. Dimension Reduction: In order to perform the face recognition (classification) task with reduced complexity and acceptable performance, usually features that are irrelevant, redundant, or noisy are excluded from the representation. This is the process of reducing the number of random variables under consideration.

iv. Data Balancing: A balanced training set leads to an equal feature space, improving recognition accuracy and fairness. This step forms the task of balancing the representation of classes.

v. Discretization: The final discretization step is the process of transforming continuous functions, models and equations into discrete forms.

The common bottleneck of these steps is the requirement for the model to facilitate a decision based on a collective representation of big data using a calculated estimation. These pre-processing steps require complex computations constrained by the number of dimensions or features to account for. As in the case of facial recognition application, the larger the dimensions, the more complex is the computation process. Hence, the main challenge faced in modelling is in translating the classical computational tasks (single instance) into a distributing computing task (MapReduce or Spark). This is an additional step that was not essential in traditional approaches. In addition, another area of concern is the accuracy or error rate of performing these tasks using distributed computing frameworks vis-à-vis classical computational task.

Data pre-processing is a critical phase in data mining. Data in the real world may not be perfectly collected or collated with the right purpose. There are many reasons for the collected big data to be dirty and examples of issues are: distorted images, lack of focus, image tag errors,

inherent error in counting or measuring devices, external factors, etc. To address these big data issues, we identify existing work in these areas and use that to guide our proposal as presented below.

i. Data Imputation - Handle missing values

Data imputation is a procedure that aims to fill in the missing values automatically. In most cases, variables are correlated with one another and therefore, we can use variables containing values to estimate the most probable value for the variable with missing values. The presence of missing values can impact the model development process. There are various reasons leading to missing values in many datasets e.g., all the information may not be available, data could be lost, certain data could be left out for a particular reason, etc. The easiest way of dealing with missing values is to simply ignore the data sample with missing values. However, this method is impractical when the number of affected data samples is large, it can introduce bias and distortion to the data.

One of the most popularly used imputation methods in image processing is based on the k-nearest neighbor (k-NN) algorithm[35,36]. The attribute average for all samples belonging to the same class provides a good estimate. Hence, we use k-NN algorithm for image data imputation in the proposed system.

ii. Feature selection - Identify outliers and smooth out noisy data

Outliers are observations that usually appear at the maximum or minimum end of a variable's possible value range, thereby skewing or distorting the distribution. If these rare, unusual, infrequent events are not of interest, outliers usually should be removed to avoid any adverse impact on the resulting model.

Cosine similarity of the k-NN has been widely applied to detect and remove any kind of class noise. For example, the well-known Edited Nearest Neighbor (ENN) [10,37] consists of removing all data samples whose class labels do not agree with majority of their k-nearest neighbors. In [38], the authors proposed a variant of the ENN, which changes the class labels of inaccurate examples. More studies of k-NN based noise filters termed as edition-based models can be found in literature [39,40].

In addition, two new approaches to remove noisy data points for big data pre-processing were proposed in [41]. The first approach called homogeneous ensemble for big data (HME-BD) uses random forest as a single base classifier. The second approach called heterogeneous ensemble for big data (HTE-BD) uses three different classifiers: k-NN, logistic regression and random forest. Using multiple algorithms implemented in the Spark framework, the study reported that HME-BD is the best performing method overall in terms of original accuracy and computing time. Further, the number of data partitions had a low impact in the performance of their noise removal process, while the voting strategy of ensemble had a high impact in the classification performance.

Considering the abovementioned previous research literature, we decided to use Open Source Computer Vision Library (OpenCV), an open-source library that includes several hundreds of computer vision algorithms to pre-process the images before submitting the spark ML pipeline. Open CV is capable of advanced image feature engineering such as changing color spaces, morphing, blending, contouring, segmentation, and deep transformations. In this work, we consider image processing aspects that are closer to face recognition such as gradient and edge transformations, foreground extraction and facial features.

iii. Data reduction - Reduce number of variables

Complex data analytics may take a very long time to run on a big data set. Data reduction can be employed to obtain a reduced representation of the data set that is much smaller in volume while capable of producing the same (or almost the same) analytical results. The two main data reduction approaches are feature selection and feature extraction.

For feature selection, several approaches have been proposed to enable data reduction techniques to pre-process big datasets. In [29,42], the authors perform feature selection on big datasets using the k-NN rule within an evolutionary mode. In [43], the authors utilized a feature-weighted version of the k-nearest neighbor algorithm for feature selection. In another work, a feature selection algorithm named BELIEF was proposed for processing big data with millions of features and implemented under the Apache Spark framework [44]. BELIEF, a distributed distance-based algorithm was developed with distances computed locally using a novel feature weighting estimation procedure to reduce the communication between partitions to improve performance by reducing process time and redundancy.

In [45], a Hadoop MapReduce solution named MRPR was designed to enable feature extraction techniques to be applied on big datasets. On the other hand, [46] proposed Random Discretization Dimensionality Reduction (RD2R) in ensembles for dimensionality reduction and random discretization based on the Apache Spark framework. They used five big datasets with very different properties to show that their novel RD2R algorithm outperforms Random Projection Random Discretization (RPRD) and Random Forest in terms of stability, prediction accuracy and effectiveness. For the prototype development of our proposed model, we adopt Spark related data reduction libraries that are DataFrame compatible for performance reasons. We also employ k-NN feature engineering algorithms for facial feature selection and grouping.

iv. Data balancing - Represent classes equally

Problems due to highly imbalanced data are more noteworthy in big data environment where huge datasets are present. In [47], an evolutionary under-sampling (EUS) method was proposed for big data classification. The method involves two MapReduce stages: the first one builds a decision tree in each map after performing EUS; and the second one classifies the test dataset using a set of decision trees. The building phase is accelerated by a windowing approach in order to speed up the under-sampling process without any loss in accuracy. The EUS balancing procedure can be performed in a guided manner by using a genetic algorithm (GA) to obtain an optimal subset of instances. With MapReduce framework, several EUS processes when run over different chunks of the training set would help to create a model (a decision tree) with each dataset, and such generated models would then get aggregated into a classifier ensemble in the reduction phase. Additionally, to apply EUS in big data problems, two levels of parallelism were proposed with: (1) MapReduce, and (2) EUS using windowing. Another work [48], building on with previous work in [47], in-memory operations within Apache Spark were employed effectively to tackle highly imbalance datasets. We employ Naïve Bayes as the ML classifier to assess the quality of solutions. We follow similar balancing pipeline design of Spark framework, but we use decision tree classifier with additional ensemble model as shown in Figure 3.

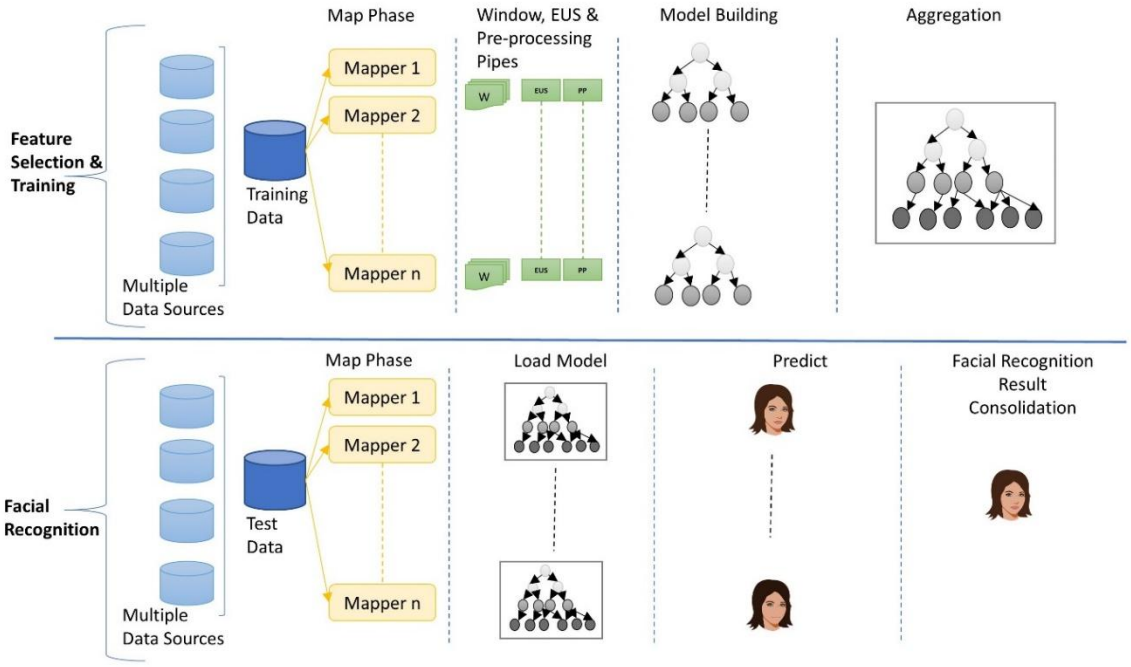

Figure 3. Decision tree classifier using EUS.

v. Discretization - Transform to discrete form

A novel Distributed Evolutionary Multivariate Discretizer (DEMD) was proposed recently using evolutionary optimization and was implemented under the Apache Spark framework [49]. By applying on several huge real-world datasets, DEMD was shown to be more accurate with simpler discretization schemes than other algorithms, including Distributed Minimum Description Length Principle (DMDLP) and Na ïve Bayes.

Overall, we provide in Table 1 a summary of our literature survey by comparing the commonly used data pre-processing methods. For each of the five pre-processing tasks described above, we identify key methods studied in literature and compare the associated parameters such as the number of features, the number of instances, the memory size and the implementation framework adopted.

Table 1. A comparison of commonly used data pre-processing methods.

Reference	Category	# Features	# Instances	Size (GB)	Framework
[35]	Missing values	73	2,534	0.0016	-
[36]	Missing values	9	1,473	0.0000	-
[38]	Outliers	8	615	0.0000	-
[39]	Outliers	11	1,025,010	0.0236	-
[40]	Outliers	11	19,020	0.0014	-
[41]	Outliers	2000	500,000	3.6000	Apache Spark
[46]	Data reduction	2000	500,000	3.6000	Apache Spark
[45]	Data reduction	41	4,856,151	1.4834	Hadoop MapReduce
[29]	Data reduction	631	65,000,000	7.4460	Hadoop MapReduce
[44]	Data reduction	631	65,000,000	7.4460	Apache Spark
[47]	Imbalance data	41	4,000,000	0.743	Hadoop MapReduce
[48]	Imbalance data	41	4,000,000	0.743	Apache Spark
[49]	Discretization	631	65,000,000	7.4460	Apache Spark

As evidenced from the summary of our literature survey given in Table 1, we observe that researchers prefer to use Apache Spark for both pre-processing and processing algorithms. Hence, we apply similar techniques for our big data processing in this study. We adopt HBase, HDFS and Spark as the BD pipeline technical stack. In conjunction to these we use special image libraries such as Open CV, MLlib, FaceNet and related deep learning algorithms that are compatible with Spark framework for the feature engineering and ML pipeline.

3.4. Big data classification

Among the popular classification techniques, we adopt the k-NN algorithm as it is most suitable for both image and text analytics as compared to other ML algorithms such as Naïve Bayes and Support Vector Machines that are biased for one data format over another. Also, the k-NN algorithm utilizes a lazy-learner approach which mimics learning by memorization, where the entire training data is loaded in feature space. The prediction is derived by comparing a new data point against the entire feature space iteratively to approximate its nearest neighbor (predicted class). This approach contrasts with eager learners' type of algorithms that constructs a model based on past observations (training data). It is also able to generalize into a single hypothesis that can cover the entire feature space for facial recognition. The key challenge when using k-NN is its resource-greedy computation due to its large memory footprint required to load and compute the entire feature space in memory. It could become computationally intensive to develop a bag-of-words model or a face recognition model that can comprehensively represent various features including a vocabulary of words for associating text-based information and other metadata usually available with facial images. Such novel considerations of multi-faceted features from various big data collections including social networks can enhance image classification. Thus, the large memory footprint problem of k-NN becomes much more compounded for big data applications such as facial recognition. In this context, among various classification algorithms, k-NN would benefit more for achieving the desired computational performance in real world applications.

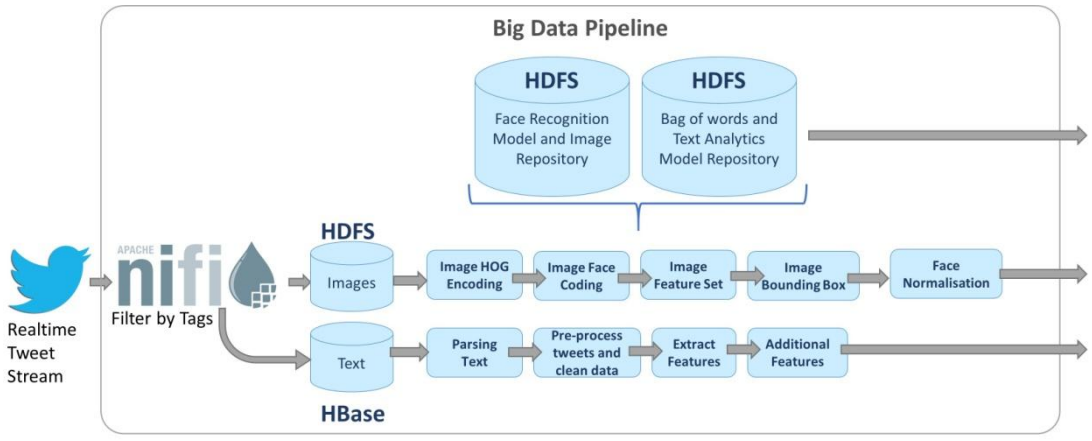

Figure 4. BD pipeline.

The BD pipeline shown in Figure 4 ingests data from various sources into a single stream and prepares for further testing in the subsequent ML pipeline. For instance, the facial recognition pre-processing needs to first be able to retrieve the region of interest (ROI), followed by detecting features using histogram of oriented gradients (HOG). Then the facial image data is coded for RGB values, and a bounding rectangle is drawn at the border for each face region detected for visualization

at the current frame. Subsequently, normalization is implemented to perceive face under various lighting conditions using histogram equalization technique. Initially, RGB image is converted to YUV color space to decouple luminance (Y) and chromatic plane (UV). Then, histogram equalization is applied to the luminance channel, Y. The formula for histogram equalization, Hk is given as follows:

$$H_k = \left[\frac{cdf_k - cdf_{min}}{t - cdf_{min}}\ (d-1)\right], k\ \in 0, \ldots d-1$$

where cdf_k is a set of cumulative density function, t is the total number of pixels for a face image, and d is the color depth at Y channel. Histogram equalization enables more facial features to be extracted. The equalized face image is then converted back to RGB image for further processing in ML pipeline.

4. Proposed classification (ML) model for facial recognition

Facial recognition technology is getting increasingly sophisticated, which requires substantial computational power and the efficiency of the algorithm adopted plays a vital role. For reliable handling and efficient processing of large-scale multimedia stream data, there is a need for a scalable, fault tolerant and loosely coupled distributed system. In the case of media that is uploaded online via social network such as Facebook or Twitter, processing for data insights is quite challenging due to the inherent massiveness in the datasets. In addition, processing for gaining insights from sentiment analysis or facial recognition is even more challenging due to their complexity of feature engineering such as facial expressions. In the past, PCA has been chosen for facial recognition to represent a larger dimensional vector of pixels that could be structured from a 2-D facial image to the compact principal components of the feature space called eigenspace projection. Further, the k-NN algorithm is applied to the spatial feature vectors for dimension reduction.

In this work, we employ Hadoop ecosystem and Spark framework due to their key features such as ease of parallelism and availability of in-memory facilities. For our facial recognition use case, we employ parallelism via MapReduce mechanism using Spark as it embraces divide-and-conquer approach for processing of large datasets successfully. In the context of our proposed model for the use case, there are two primary design goals:

1. To perform real-time ingestion of images data in distributed file system such as HDFS.
2. To support image analytics by augmenting sentimental analytics from the collected social media text information, thereby observing emotional similarities between the image and text content.

Based on reported literature, we observe that (a) the feature extraction performance and the processing time for image classification are determined by the number of working nodes in the k-NN classifier, (b) the facial expression recognition rate has slight impact on PCA, and (c) the recognition rate with different PCA and Mahalanobis distance strategy are able to produce excellent result for neutral facial expressions. However, extreme lighting and covered face continues to be a challenge for PCA based facial recognition algorithms. We describe three main classification approaches, namely k-NN, k-means clustering and decision tree implemented for the use case in this work.

4.1. K-Nearest Neighbors (k-NN)

A k-NN classifier is a slow and lazy learner as it uses the entire training set each time when

testing an example is performed. When training a k-NN classifier, it exhibits a time complexity of $O(1)$. When classifying a new instance over a training set with m instances and n attributes, it exhibits a time complexity of $O(mn + m\log(m))$, where $O(mn)$ is the time complexity for calculating the cosine-similarity between the test examples with the entire training set examples, and $O(m\log(m))$ is the time complexity for sorting the similarity when finding the k-nearest neighbors of the new example. In summary, with the increase in data size, the processing time of k-NN will increase exponentially.

A Hadoop distributed processing with a MapReduce implementation of a k-NN classifier (MR-KNN) was proposed by mapping the training examples, followed by reducing the number of examples that are nearest to the test example [50]. The MR-KNN was reported to achieve a speed-up from 16 to 149 times when tested on a dataset with 1 million examples. The speed-up of the proposed implementation depended mainly on the number of maps and k neighbors used. These results were further improved by using an Iterative Spark implementation (kNN-IS), similar to MR-KNN framework, but using multiple reducers to speed up the result aggregation process [51].

4.2. K-Means clustering

One of the simplest ML techniques for splitting a dataset is k-means, a typical non-hierarchical clustering algorithm. Its goal is to divide the sample into predetermined number (k) of non-overlapping clusters so that clusters are as homogeneous as possible with respect to the measurements used. The k-means algorithm is very efficient and perhaps the fastest clustering algorithm that can handle both long (many records) and wide datasets (many input fields). Based on clustering the training set using k-means clustering algorithm, a recent work proposed using landmark spectral clustering (LC-KNN) to increase the speed of k-NN [52]. After narrowing down the most relevant cluster, sequential k-NN was applied from a smaller set of examples to the test example. LC-KNN was evaluated on nine huge datasets showing reasonable approximation. Another study proposed cluster augmentation process (cKNN) to accelerate the speed of the k-NN [53]. The reported average accuracy for various datasets was in the range from 83% to 90%, depending on the number of clusters used. In addition, these results were improved when deep neural networks (DNN) were used to learn representative features for classification.

4.3. Decision tree

To solve the speed limitations in big data classification approaches using k-NN, we implement decision tree classifiers. A recent study had proposed two multivariate decision tree classifiers for dealing with large datasets [54]. The first classifier employed a random partition, and the second classifier employed PCA-partitioned method. Fully balanced binary trees were generated based on multivariate combination of weights and by adopting a median-based method to select the divide value. Many research studies conducted with big datasets have used a ‘divide and conquer’ algorithmic principle of decision trees successfully. Such a decision tree approach requires clustering, splitting, or partitioning the data to a manageable size that can then be used by the classifier.

5. Prototype implementation and results

Recent advances in Social Networking (SN), Internet of Things (IoT), crowdsourcing and

cloud-based services have brought techniques and technologies of facial recognition, ML and big data to converge. For our facial recognition use case, we implemented an end-to-end prototype solution to ingest, process and analyze image-based big datasets using our proposed big data architecture (Figure 2). Next, we describe the BD pipeline and ML pipeline components for the implementation of the end-to-end facial recognition prototype.

Stage 1: Big Data Collection

Massive collections of facial images are available in various public datasets and SN hubs such as Facebook. We make use of public data sets from Facebook and Twitter to evaluate different techniques in facial analysis. Our prototype solution caters to data acquisition of facial images from such big data streams using web crawlers as well as from reliable data sources. The three main types of objects returned by Twitter used in our solution are described in Table 2. For our prototype development, we employ the latest beta version of v2 API of Twitter, which supports new features such as choosing specific fields to receive data within the response and streaming rules that allows changes without dropping connections.

Stage 2: Big Data Source

In this study, Twitter APIs with schema shown in Table 2 are used as the prime data source, media is used for facial recognition and tweet for sentiment processing, testing of text and image analytics integration, validating our integrated prototype for insight engineering. We use acquire live data for testing our prototype with tags and geofence. To overcome Twitter API quota restrictions for developers, we created two functions. The first function was designed to receive ~1% of all public Tweets in real-time based on specific ruleset, e.g. monitoring data based on certain Twitter user accounts or keywords. The second function was used for code testing and demonstrations since it allowed controlled incoming tweet data. A variety of features such as face tagging and geofencing were used in novel ways to select the dataset from real-time streams.

Table 2. Tweet data set.

Object	Description
Tweet	The Tweet object has a long list of root-level fields, such as id (unique ID for each tweet), text (tweet content), and created at (date timestamp of tweet). Tweet objects are also the parent object to several child objects including user and media objects.
User	The user object contains Twitter user account metadata describing the referenced user. Fields included are name, username, date of creation of account, number of followers, tweet counts and more.
Media	If a Tweet contains media (such as images), then the media object can be requested using the media.fields parameter and includes fields such as the media_key, type and URL.

Stage 3: Ingestion

For implementing the Ingestion phase of the BD pipeline, we employed Apache Nifi and minifi to automate the process of ingesting data from the staging environment into Hadoop HDFS

seamlessly. We describe below two main steps for implementing the Ingestion of big data of the facial images in the pre-processing phase:

i. Ingestion Step 1: In this step, Apache Nifi listens to the staging folders as shown in Figure 5. We developed Python scripts to perform data cleaning, data enrichment, and extraction of key attributes as well as to integrate multiple disparate source files of facial image data from the staging environment into a single dataset. We make use of Apache Nifi to execute these scripts to pre-process data and to store the output into an intermediate folder prior to transferring them into HDFS.

Figure 5. Implementation of ingestion stage 1 of facial image data.

ii. Ingestion Step 2: This step creates an additional Nifi workflow to listen to new files that require to be transferred into HDFS as shown in Figure 6. When a new file arrives, Nifi will perform a PUT command to move data from the host environment into Hadoop HDFS via a Web HDFS interface.

Figure 6. Implementation of ingestion stage 2 of facial image data.

Stage 4: Big Data Storage

For this prototype implementation, HDFS was used as the file storage to store the images captured from the public tweets, and to reference image uploads of the target for facial recognition. To maintain within the storage limit, the storing and retrieval of images was constrained to a maximum of 5 images for each run. HDFS was also used to facilitate as a host for images utilised on the Tableau dashboard. After the data ingestion, for the pipeline processing, we had stored various image and text separately for further processing. These data were then placed in a staging environment (specific folders on local host) before they were ingested into HDFS.

We made use of Cloudera cluster environment to implement our proposed facial recognition prototype. The main storage choice adopted was HDFS, and we employed Spark pipelines to deliver our proposed solution in a single node cluster. In addition, we employed Apache Ambari tool to monitor the Hadoop system resources and their performance. An example Ambari monitoring dashboard is shown in Figure 7. Such a dashboard is useful to manage/configure the Hadoop cluster based on the resources required for large amounts of facial image data that gets ingested from various big data sources into the system.

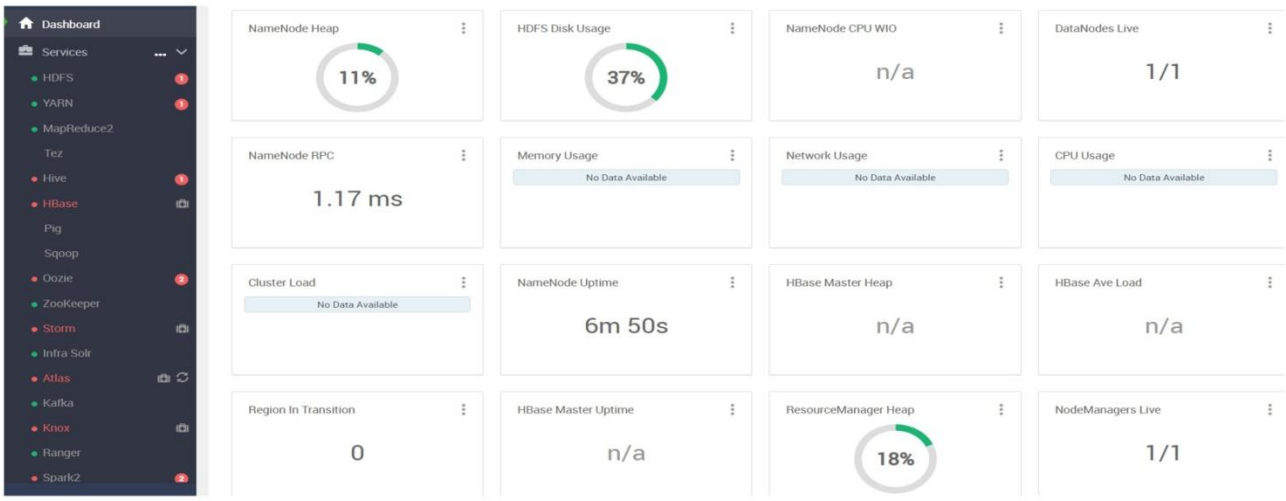

Figure 7. System performance monitoring dashboard.

Stage 5: ML Process and Analysis

The image dataset for facial recognition was captured by considering the real-world challenges such as variations in pose, illumination, expressions, background, etc. We obtained the training data using our own personal images that had been suitably anonymized. We used two sets of training data and one set of test data. For the first training dataset, we considered mostly frontal color images, with only one image per user. The second training data was the live real-time tweet stream feeds that was acquired from the big data processing pipeline as explained earlier. The cleansed data set consisted of multiple real-world images per user as shown in Figure 8.

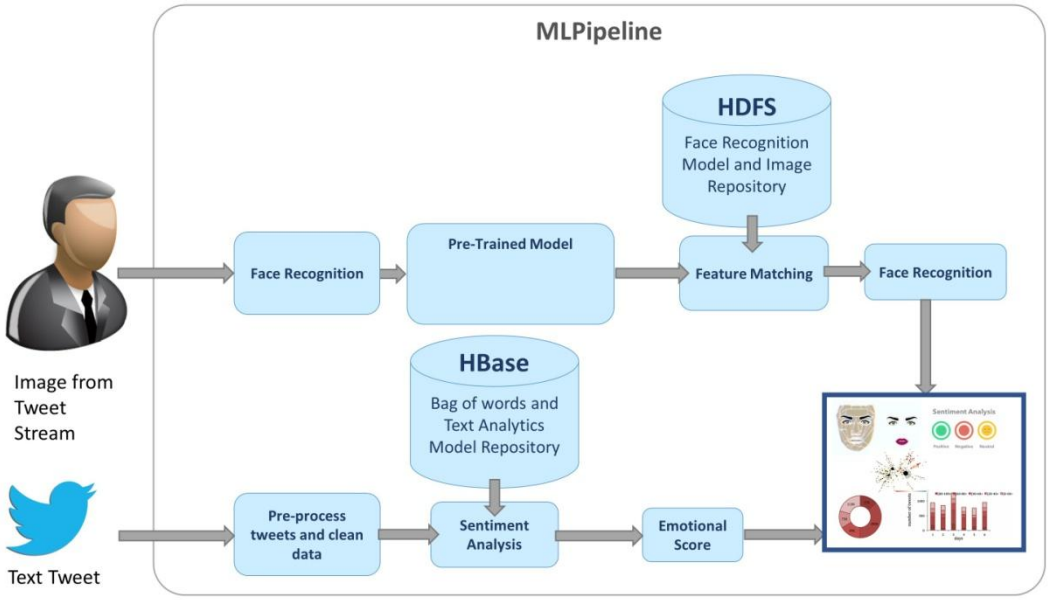

Figure 8. Training and testing data set in ML pipeline.

For the actual ML pipeline proposed, our key solution focus was on its implementation using Spark. We invoked Spark's Pipelines API and customized our programs to automate the ML tasks required for the image-based feature process and analysis stage of implementation. We created a ML

workflow by integrating the Spark Transformers and Estimators as a sequence of pipeline programs using Spark ML. While a Transformer includes feature transformation methods and learned models, an Estimator includes learning model fitting and training on the pre-processed facial data. Figure 9, Workflow process and analysis of facial images in ML pipeline - provides a workflow of the process and analysis phase in the ML Pipeline for the facial recognition use case.

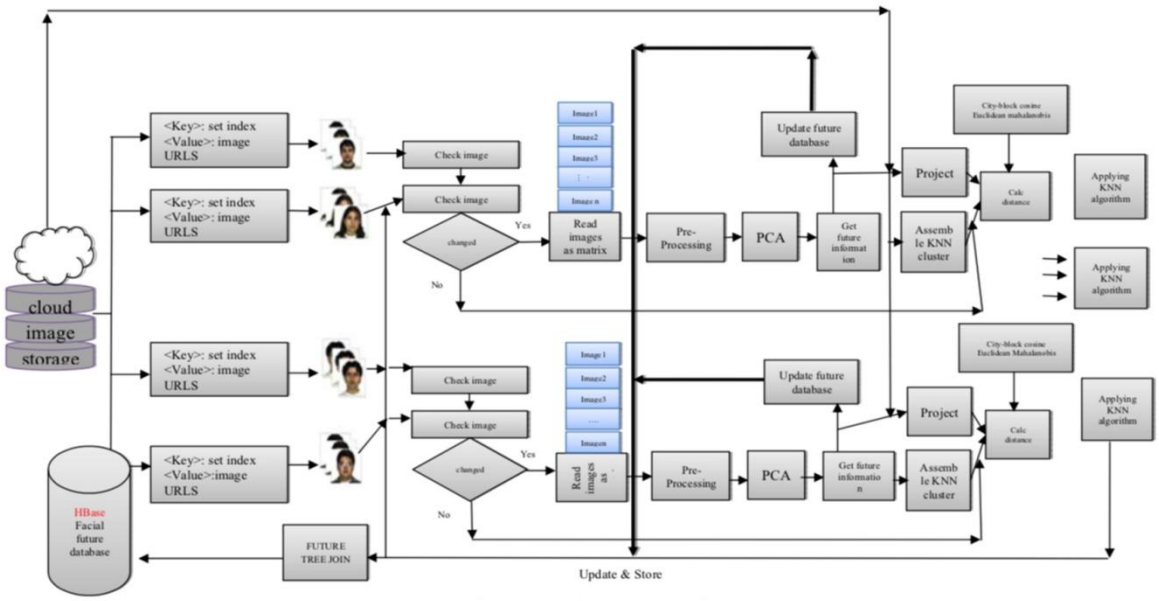

Figure 9. Workflow process and analysis of facial images in ML pipeline.

In general, the facial recognition implementation in Spark ML Scala builds a ML model in an ordered sequence of processes as follows: (1) Feature extraction, transformation and selection. (2) Predictive model training a based on these vectors and label. (3) Prediction using the generated model. (4) Evaluating the model (performance and accuracy). Spark `MLib` provides two top level abstractions to facilitate the development of this pipeline: `transformers` and `estimators`. A transformer implements a method `transform()` which will convert one `DataFrame` into another, generally appending one or more new column. For example, a transformer will take all the columns as features of each entry on the Data Frame and map it into a new column (feature vectors). The estimator will be responsible for applying the learning algorithm that fits or trains on data. It implements the method `fit()` that accepts a `DataFrame,` and produces a Model that is a transformer. We achieve image feature extraction using six selected features such as tags, face features, color, metadata and image features including labels. Batch training was conducted using `k-means` clustering algorithm (Figure 10).

```
// Run Pipeline
val k = 3
val kmeans = new KMeans().setK(k).setSeed(1).setFeaturesCol("features")
val kmeans_model = kmeans.fit(va_transformed_df)
val va_cluster_df = kmeans_model.transform(va_transformed_df)
      .select("feature1", "feature2", "feature3", "feature4", "feature5", "feature6","prediction")
```

Figure 10. Sample spark ML code snippet for ML pipeline with k-means fit.

Real-time stream processing supported by Spark ML and Discrete Streaming techniques were employed to extract information from unbounded video stream data and to divide them into smaller chunks of image data. We implemented k-NN algorithm on Apache Spark platform using a hybrid spill tree approach to achieve high accuracy and search efficiency. The simplicity of k-NN and the lack of tuning parameters makes k-NN a useful baseline model for many ML problems, and is a best-fit for our study. Apache Spark consists of multiple layers of nodes and each layer of the k-NN classifier can be fully connected to the next layer in the network. For our prototype, nodes in the input layer were used to represent the input data. All other nodes mapped inputs to outputs by a linear combination of the inputs with the node's weights w and bias b and by applying an activation function. This can be written in matrix form with K+1 layers as follows:

$$y(x) = f_k(\ldots f_2\ (w_2^T f_1(w_2^T x + b_1) + b_2)\ldots + b_k)$$

For the nodes in intermediate layers, we used the sigmoid (logistic) function:

$$f(z_i) = \frac{1}{1 + e^i}$$

For the nodes in the output layer, we used the softmax function:

$$f(z_i) = \frac{e^{z_k}}{\sum_{k=1}^{N} e^{z_k}}$$

The number of nodes *N* in the output layer corresponds to the number of classes. Spark ML employs backpropagation for learning the model. We adopted factorization machines that can estimate interactions between features even in problems with huge sparsity such as the facial recognition problem of our use case. The `spark.ml` implementation supports factorization machines for classification. The formula used for factorization machines is given below:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle\, x_i x_j$$

The first two terms denote the intercept and linear term, and the last term denotes pairwise interactions term. The pair $\langle v_i, v_j \rangle$ describes the i^{th} variable with k factors. We employed factorization machines for establishing optimization criterion (logistic loss) with mean square error. Factorization machines can be used for classification through sigmoid function. To demonstrate our ML pipeline code, in Figure 11 we provide a sample code in SparkML for the image grabbing function (from a real time streaming source) and vector assembly of images with the selected features extracted from facial images of various users. When the functions are declared, we chain them up into a single pipeline. This pipeline in turn can be used to fit the training data automatically. This had helped to reduce the latency in training as the pre-processing and training functions were performed continuously in-memory as compared to processing the data in a step-by-step manner which would incur some cost in loading each processed instance in-memory after completing each ML step.

```
val grabber = new OpenCVFrameGrabber(0)
grabber.start()
val initialImage = grabber.grab()
val canvasFrame = new CanvasFrame("Source1")
canvasFrame.setCanvasSize(initialImage.imageWidth / 2, initialImage.imageHeight / 2)
grabber.setFrameRate(grabber.getFrameRate)
Loader.load(classOf[opencv_objdetect])
val faceCascade = new CascadeClassifier(classifier(args).getAbsolutePath)
val features_col = Array("feature1", "feature2", "feature3", "feature4", "feature5", "feature6")
val vector_assembler = new VectorAssembler()
      .setInputCols(features_col)
      .setOutputCol("features")
val va_transformed_df = vector_assembler.transform(final_image_df)
```

Figure 11. Sample Spark ML code snippet for image grab and vector assembler.

In addition to ML, we used Spark to create logical views of the dataset for various queries to perform further analytics. Figure 12 gives sample code to demonstrate the image view creation.

```
// Image View Creation
val width = img.getWidth
val height = img.getHeight
// create new image of the same size
val out = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR)
for (j <- 0 until image_array.size - 1) {
    val color_tuple = colors(image_array(j)._3)
    out.setRGB(image_array(j)._1, image_array(j)._2,
      new Color(color_tuple._3, color_tuple._2, color_tuple._1).getRGB)
}
```

Figure 12. Image view creation.

Stage 6: Visualization

We developed the final visualization stage of our big data architecture to provide visual outputs of data analytics performed for the facial recognition system. We employed Apache Zeppelin built-in Apache Spark integration tools to perform the data visualization tasks. The analysis and interpretation of the processed data was directly extracted from HDFS. The visualization layer comprised of two key components as summarized below:

i. Business Intelligence Dashboards - The logical views created from Stage 4 that were queried using Spark SQL were visually made available as business intelligence dashboards using a variety of dynamic graphical charting features. The tableau dashboards generated with a test use case data for images and tweet texts are shown in Figure 13 for facial recognition and Figure 14 for sentiment analysis of tweets.

ii. Machine Learning Evaluation - The results of the Sentiment Analysis model were also visualized to determine an evaluation of the ML model developed for the facial recognition prototype. Several metrics were used to evaluate our ML classifier implemented for the use case of our big data architecture. A visualization of one such metric is given in Figure 15 as an example. It shows that an area under curve (AUC) score of 0.952 was achieved under a receiver operating characteristic curve (ROC) that was generated for evaluating the facial image classification of our prototype. In ML, the ROC curve is obtained by plotting the true positive

rate (TPR) against the false positive rate (FPR) at various threshold settings, where TPR denotes the probability of correct facial detection and FPR denotes the probability of false detection. An AUC value close to 1 shows a high performance of the classifier.

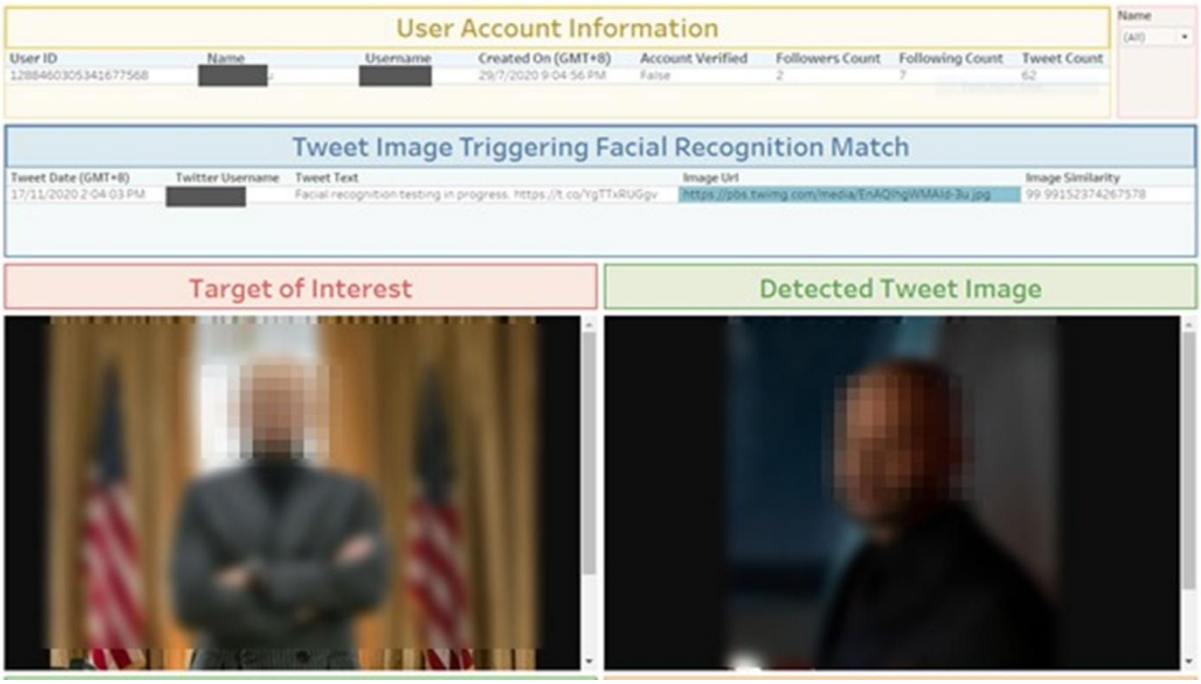

Figure 13. Face recognition dashboard (image blurred to protect privacy).

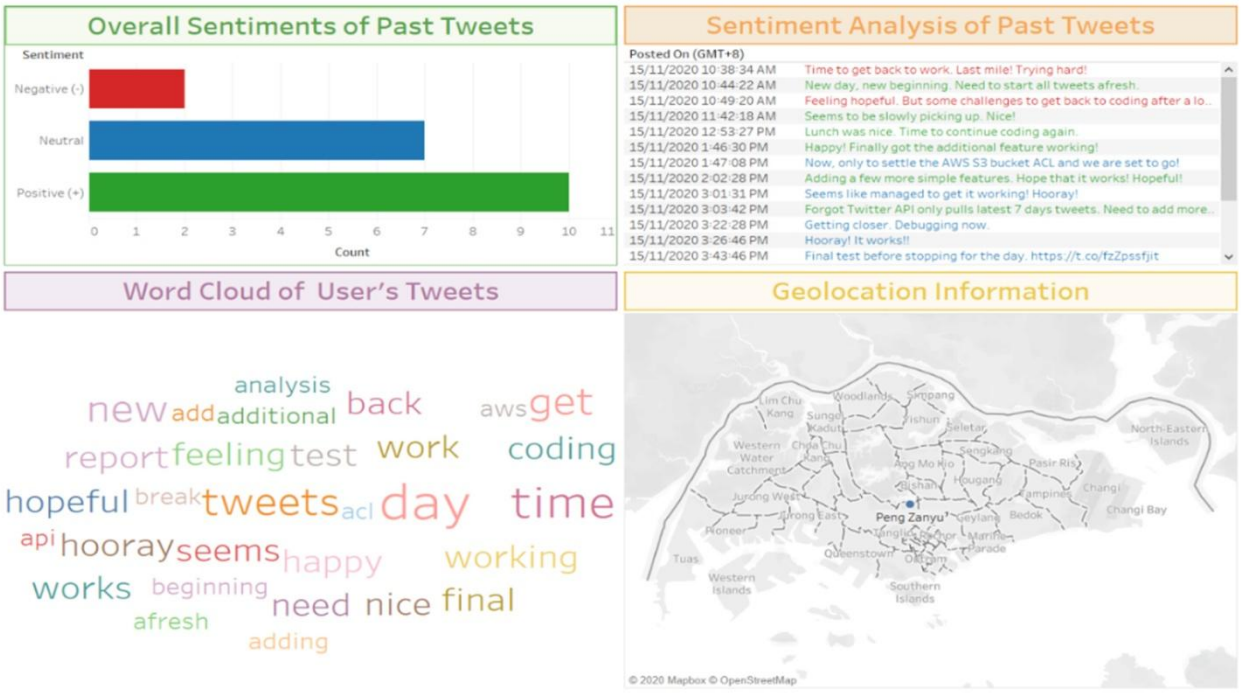

Figure 14. Tweet dashboard.

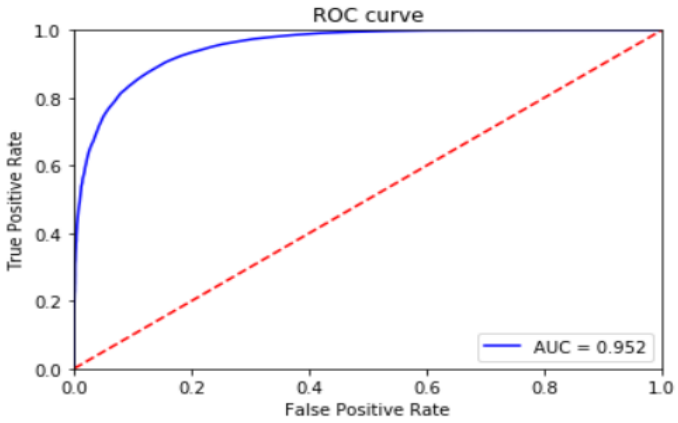

Figure 15. Sample ROC curve.

Result Discussion

When retrieving the past seven days' worth of tweets via Twitter API from a user, the data was received as JSON objects and converted into a Spark DataFrame, which required the processing to be scalable and fast. This is essential because there may be a large number of tweets, and also there may be multiple users tweeting the same matching images. Each return from Twitter consisted of only 10 entries per "page", thus we had to iterate through the "pages" of the tweet entries until all tweets were retrieved. Thereafter, the different returns were merged into one Spark DataFrame. To manage the large data set for face recognition, the prototype was scoped into a particular geofence to restrict the search space. We observe from the returned tweets that a user may include geolocation information in some tweets and if not properly handled, it can cause an exception as different pages of tweet entries will have a different number of fields. As such, there was a need to check if geolocation information could be found in that particular page of tweet entries and if so, to be parsed accordingly. The above scoping constraints built into the script helped us to have control on the dataset loaded in memory at any point of the prototype development.

As for text tweets that were relatively short, only basic data cleaning was required. Punctuation was removed and the text was converted to lowercase. Thereafter, tokenisation and removal of stop words were performed via the spark.ml library functions (Tokenizer and StopWordsRemover). Afterwards, words less than three or more than thirteen characters are removed. The result is a Spark DataFrame containing word counts, which would eventually be used to generate word cloud in Tableau. Separately, sentiment analysis is performed on each tweet and the sentiment scores were tagged to it. This was done using TextBlob. The sentiment scores were then classified into three categories, namely positive, neutral, and negative. Total counts of the tagged tweets were also stored as these would serve as useful summaries (descriptive analytics) for basic profiling. The integrated dashboard as shown in Figure 14 provides an illustration of the visualization of data insights from tweet sentiment analysis.

6. Conclusion

The use of big data technologies to store, process and analyze data has changed the context of knowledge discovery from data from various sources including social networks. This work intersects

three areas, big data, machine learning and facial recognition augmented with sentiment analysis of social engineering data as a modest initial step in this direction. This was achieved by proposing a novel big data architecture consisting of BD pipeline and ML pipeline and its use case for facial recognition application was demonstrated by implementing the prototype.

Real-time stream processing supported by Spark ML and Discrete Streaming techniques were employed to extract information from unbounded video stream data into smaller chunks of image data. In particular, the data pre-processing and data mining tasks that were previously implemented in a single computation node have become suboptimal with big data warranting real-time distributed computing. ML models in distributed nodes are required to be re-developed by looking at only a subset of the observation space. To achieve this, we proposed a novel big data architecture that can operate optimally on images and text to augment insights on person's mood/sentiments. Further, the big data pre-processing methods and classification models adopted were guided by the 3V tenets of big data (Volume, Velocity and Variety). We demonstrated the application of our proposed big data architecture using a facial recognition and sentiment analysis integrated pipeline. We employed Hadoop and Spark frameworks and the associated ML libraries for open source streaming conversion and image pre-processing. We highlighted key implementation components and visualization of data analytics performed for the facial recognition use case. Further, our use case development and observations during implementation provide domain-specific future research areas that can be explored in big data environments.

This paper presented a prototype pipeline using open-source image and text processing tools for rapid acquisition and visualization of a stream-specific dataset in the immediate aftermath of image conversion window with relevant pattern recognition algorithms. For the purpose of demonstrating our proposed architecture in this research work, the prototype pipeline is implemented to use a small number of interactively labeled real time data samples from open image and tweet data sets, and an active learning framework, coupled with unsupervised open grabber tool embedding. This facilitated to obtain a relevant corpus without extensive labeling or feature engineering effort. Using a more supervised, interactive and lightweight system, images from a large video corpus containing millions of unlabeled image frames can be employed and would be more helpful in selecting key image frames that access the facial recognition system. As an extension, in future we could perform comparison study using various cloud facilitated image recognition products such as AWS SageMaker, Rekognition and Cloud Vision, and benchmark against FaceNet and Open CV libraries.

Finally, facial recognition technology, powered with big data and real-time stream processing abilities can help us solve difficult problems, increase security, and improve stakeholder interaction experience. However, the technology could be misused resulting in privacy invasion and security breaches. Hence, we recommend to advocate for strong digital ethics from perspectives of fairness, inclusiveness, accountability, trust and adaptability to be incorporated into the framework for future research.

Acknowledgements

The authors acknowledge and thank Twitter for allowing the use of Twitter Developer Kit V2 and public tweets with specific taglines and dates, including images in the prototype development of this work. The authors wish to thank the student team (Koay Chin Yang, Wu Hua Qing, Woo Chia Wei, Quan Yu, Meng Yang, Tan Boon Leong, Wang Jing) for the assistance provided towards the data cleansing, algorithmic comparison study and sentiment analysis implementation. The authors also wish to thank another student team (Chaitanya Muthaiyan, Nathan Timothy Handoko, Noel Situ

Wen, Peng Zanyu) for their participation towards the prototype dashboard development. Finally, the authors wish to acknowledge the invaluable inputs provided by the reviewers and editor that helped in enhancing the quality of the paper.

Conflict of interest

The authors declare that there is no conflict of interest.

References

1. Chen M, Mao S, Liu Y (2014) Big data: A survey. *Mobile networks and applications* 19: 171–209.
2. McAfee A, Brynjolfsson E, Davenport TH, et al. (2012) Big data: the management revolution. *Harvard business review* 90: 60–68.
3. Venkatraman R, Venkatraman S (2019) Big Data Infrastructure, Data Visualisation and Challenges. *Proceedings of the 3rd International Conference on Big Data and Internet of Things*, 13–17.
4. Labrinidis A, Jagadish HV (2012) Challenges and opportunities with big data. *Proceedings of the VLDB Endowment* 5: 2032–2033.
5. Venkatraman S, Venkatraman R (2019) Big data security challenges and strategies. *AIMS MATHEMATICS* 4: 860–879.
6. Masi I, Wu Y, Hassner T, et al. (2018) Deep face recognition: A survey. *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, 471–478.
7. Singh A, Bhadani R (2020) *Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter*. Packt Publishing.
8. Zhu Y, Jiang Y (2020) Optimization of face recognition algorithm based on deep learning multi feature fusion driven by big data. *Image Vision Comput* 104: 104023.
9. Reddy KS, Krishna VV, Kumar VV (2016) A Method for Facial Recognition Based On Local Features. *International Journal of Mathematics and Computation* 27: 98–109.
10. Qateef JS, Kazm AA (2016) Facial expression recognition via mapreduce assisted k-nearest neighbor algorithm. *International Journal of Computer Science and Information Security* 14: 170.
11. Sirovich L, Kirby M (1987) Low-dimensional procedure for the characterization of human faces. *Josa a* 4: 519–524.
12. Turk MA, Pentland AP (1991) Face recognition using eigenfaces. *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, 586–587. IEEE Computer Society.
13. Bruce V, Young A (1986) Understanding face recognition. *British journal of psychology* 77: 305–327.
14. Parkhi OM, Vedaldi A, Zisserman A (2015) Deep face recognition.
15. He X, Yan S, Hu Y, et al. (2005) Face recognition using Laplacianfaces. *IEEE T Pattern Anal* 27: 328–340.
16. Deng J, Guo J, Xue N, et al. (2019) Arcface: Additive angular margin loss for deep face recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4690–4699.

17. Zhou E, Cao Z, Yin Q (2015) Naive-deep face recognition: Touching the limit of LFW benchmark or not? *arXiv preprint arXiv:150104690*.
18. Wang H, Wang Y, Zhou Z, et al. (2018) Cosface: Large margin cosine loss for deep face recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5265–5274.
19. Wen Y, Zhang K, Li Z, et al. (2016) A discriminative feature learning approach for deep face recognition. *European conference on computer vision*, 499–515.
20. Deng J, Zhou Y, Zafeiriou S (2017) Marginal loss for deep face recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 60–68.
21. Ding H, Zhou SK, Chellappa R (2017) Facenet2expnet: Regularizing a deep face recognition net for expression recognition. *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, 118–126.
22. Wang F, Chen L, Li C, et al. (2018) The devil of face recognition is in the noise. *Proceedings of the European Conference on Computer Vision (ECCV)*, 765–780.
23. Benhlima L (2018) Big data management for healthcare systems: architecture, requirements, and implementation. *Advances in bioinformatics* 2018.
24. Asaithambi SPR, Venkatraman R, Venkatraman S (2020) MOBDA: Microservice-Oriented Big Data Architecture for Smart City Transport Systems. *Big Data and Cognitive Computing* 4: 17.
25. Costa C, Santos MY (2016) BASIS: A big data architecture for smart cities. *2016 SAI Computing Conference (SAI)*, 1247–1256.
26. He X, Wang K, Huang H, et al. (2018) QoE-driven big data architecture for smart city. *IEEE Commun Mag* 56: 88–93.
27. Lopez D, Manogaran G (2016) Big data architecture for climate change and disease dynamics. *The human element of big data: issues, analytics, and performance*, 301–331.
28. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51: 107–113.
29. Peralta D, Del R ó S, Ram írez-Gallego S, et al. (2015) Evolutionary feature selection for big data classification: A mapreduce approach. *Math Probl Eng* 2015.
30. Gao W, Zhao X, Gao Z, et al. (2019) 3D Face Reconstruction From Volumes of Videos Using a Mapreduce Framework. *IEEE Access* 7: 165559–165570.
31. Mahmoud SM, Habeeb RS (2019) Analysis of Large Set of Images Using MapReduce Framework. *International Journal of Modern Education and Computer Science* 11: 47.
32. Apache Spark™. A unified analytics engine for large-scale data processing.
33. Hazarika AV, Ram GJSR, Jain E (2017) Performance comparision of Hadoop and spark engine. *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, 671–674.
34. Zaharia M, Chowdhury M, Das T, et al. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 15–28.
35. Luengo J, Garc á S, Herrera F (2012) On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowl Inf Syst* 32: 77–108.
36. Batista GE, Monard MC (2003) An analysis of four missing data treatment methods for supervised learning. *Appl Artif Intell* 17: 519–533.
37. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. *IEEE T Syst Man Cy B*, 408–421.

38. Sánchez JS, Barandela R, Marqués AI, et al. (2003) Analysis of new techniques to obtain quality training sets. *Pattern Recogn Lett* 24: 1015–1022.
39. Garcia S, Derrac J, Cano J, et al. (2012) Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE T Pattern Anal* 34: 417–435.
40. Triguero I, Derrac J, Garcia S, et al. (2011) A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE T Syst Man Cy C* 42: 86–100.
41. García-Gil D, Luengo J, García S, et al. (2019) Enabling smart data: noise filtering in big data classification. *Inform Sciences* 479: 135–152.
42. Xue B, Zhang M, Browne WN, et al. (2015) A survey on evolutionary computation approaches to feature selection. *IEEE T Evolut Comput* 20: 606–626.
43. Navot A, Shpigelman L, Tishby N, et al. (2005) Nearest neighbor based feature selection for regression and its application to neural activity. *Advances in neural information processing systems* 18: 996–1002.
44. Ramírez-Gallego S, García S, Xiong N, et al. (2018) BELIEF: A distance-based redundancy-proof feature selection method for Big Data. *arXiv preprint arXiv:180405774*.
45. Triguero I, Peralta D, Bacardit J, et al. (2015) MRPR: a MapReduce solution for prototype reduction in big data classification. *Neurocomputing* 150: 331–345.
46. García-Gil D, Ramírez-Gallego S, García S, et al. (2018) On the Use of Random Discretization and Dimensionality Reduction in Ensembles for Big Data. *International Conference on Hybrid Artificial Intelligence Systems*, 15–26.
47. Triguero I, Galar M, Vluymans S, et al. (2015) Evolutionary undersampling for imbalanced big data classification. *2015 IEEE Congress on Evolutionary Computation (CEC)*, 715–722.
48. Triguero I, Galar M, Merino D, et al. (2016) Evolutionary undersampling for extremely imbalanced big data classification under apache spark. *2016 IEEE Congress on Evolutionary Computation (CEC)*, 640–647.
49. Ramírez-Gallego S, García S, Benítez JM, et al. (2018) A distributed evolutionary multivariate discretizer for big data processing on apache spark. *Swarm Evol Comput* 38: 240–250.
50. Maillo J, Triguero I, Herrera F (2015) A mapreduce-based k-nearest neighbor approach for big data classification. *2015 IEEE Trustcom/BigDataSE/ISPA* 2: 167–172.
51. Maillo J, Ramírez S, Triguero I, et al. (2017) kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowl-Based Syst* 117: 3–15.
52. Deng Z, Zhu X, Cheng D, et al. (2016) Efficient kNN classification algorithm for big data. *Neurocomputing* 195: 143–148.
53. Gallego A-J, Calvo-Zaragoza J, Valero-Mas JJ, et al. (2018) Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation. *Pattern Recogn* 74: 531–543.
54. Wang F, Wang Q, Nie F, et al. (2018) Efficient tree classifiers for large scale datasets. *Neurocomputing* 284: 70–79.