AUTHOR'S PROOF

Metadata of the article that will be visualized in OnlineFirst

1	Article Title	**Preprocessing framework for scholarly big data management**	
2	Article Sub- Title		
3	Article Copyright - Year	**The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022 (This will be the copyright line in the final PDF)**	
4	Journal Name	Multimedia Tools and Applications	
5	Corresponding Author	Family Name	**Khan**
6		Particle	
7		Given Name	**Samiya**
8		Suffix	
9		Organization	Jamia Millia Islamia
10		Division	
11		Address	New Delhi, India
12		e-mail	samiyashaukat@yahoo.com
13	Author	Family Name	**Alam**
14		Particle	
15		Given Name	**Mansaf**
16		Suffix	
17		Organization	Jamia Millia Islamia
18		Division	
19		Address	New Delhi, India
20		e-mail	malam2@jmi.ac.in
21	Schedule	Received	17 November 2020
22		Revised	3 August 2021
23		Accepted	13 July 2022
24	Abstract	Big data technologies have found applications in disparate domains. One of the largest sources of textual big data is scientific documents and papers. Scholarly big data has been used in numerous ways to develop innovative applications such as collaborator discovery, expert finding and research management systems. With the evolution of machine and deep learning techniques, the efficacy of such applications has risen manifold. However, the biggest challenge in the development of deep learning models for scholarly applications in cloud-based environment is the under-utilization of resources because of the excessive time required for textual preprocessing. This paper presents a preprocessing pipeline that uses Spark for data ingestion and Spark ML for performing preprocessing tasks. The proposed approach is evaluated with the help of a case study, which uses LSTM-based text summarization to generate title or summaries from abstracts of scholarly articles. Results indicate	

AUTHOR'S PROOF

		a substantial reduction in ingestion, preprocessing and cumulative time for the proposed approach, which shall manifest reduction in development time and costs as well.
25	Keywords separated by ' - '	Deep learning applications - Preprocessing pipeline - Scholarly big data - Scholarly data applications - Spark ML
26	Foot note information	Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

JrnlID 11042_ArtID 13513_Proof#1 - 30/07/2022

Multimedia Tools and Applications
https://doi.org/10.1007/s11042-022-13513-8

1216: INTELLIGENT AND SUSTAINABLE TECHNIQUES FOR MULTIMEDIA BIG DATA MANAGEMENT FOR SMART CITIES SERVICES

Preprocessing framework for scholarly big data management

Samiya Khan[1] · **Mansaf Alam**[1]

Received: 17 November 2020 / Revised: 3 August 2021 / Accepted: 13 July 2022

Abstract
Big data technologies have found applications in disparate domains. One of the largest sources of textual big data is scientific documents and papers. Scholarly big data has been used in numerous ways to develop innovative applications such as collaborator discovery, expert finding and research management systems. With the evolution of machine and deep learning techniques, the efficacy of such applications has risen manifold. However, the biggest challenge in the development of deep learning models for scholarly applications in cloud-based environment is the under-utilization of resources because of the excessive time required for textual preprocessing. This paper presents a preprocessing pipeline that uses Spark for data ingestion and Spark ML for performing preprocessing tasks. The proposed approach is evaluated with the help of a case study, which uses LSTM-based text summarization to generate title or summaries from abstracts of scholarly articles. Results indicate a substantial reduction in ingestion, preprocessing and cumulative time for the proposed approach, which shall manifest reduction in development time and costs as well.

Keywords Deep learning applications · Preprocessing pipeline · Scholarly big data · Scholarly data applications · Spark ML

1 Introduction

Artificial intelligence has revolutionized many domains by providing a technological platform for development of innovative applications and use cases. Scholarly applications is one such field that makes extensive use of Natural Language Processing (NLP) in the backend along with machine learning and deep learning to develop innovative applications for

✉ Samiya Khan
samiyashaukat@yahoo.com

Mansaf Alam
malam2@jmi.ac.in

Q1

[1] Jamia Millia Islamia, New Delhi, India

researchers [41]. From recommender systems [3] to text summarization mechanisms [12], the implementation of complex analytical techniques on big scholarly data can be used to solve different problems to achieve the common benefit of the research community.

A classic example of this assertion is automatic keyword extraction [29], which extracts keywords from scholarly articles using multiple parameters, such as the frequency occurrence of keywords [6], citations [33], author relationships, and others. In addition, research paper recommenders [20], venue recommendation systems [35] and analytically-enabled research management systems [16] are some other existing applications of big scholarly data analytics.

Big scholarly data comprise of textual and image data. Different applications are expected to use big scholarly data in different ways. For instance, if a research article writing system offers image caption generation as a service, then the images of the scholarly article are used. In this regard, a majority of scholarly applications focus on textual data analytics, for which the required text is extracted from a PDF or HTML webpage, cleaned using NLP algorithms [7] and used as an input to create diverse machine learning and deep learning applications.

The problem with development of deep learning models for natural language processing-based scholarly applications is that the preprocessing stage is extremely resource-intensive, and time-consuming. Moreover, with any deep learning application, the accuracy of result depends on the amount of data used for training [14]. However, as dataset size increases for NLP-based applications, preprocessing stage of the deep learning model development becomes extremely resource intensive.

Considering the complex nature of the application, a model can only be appropriately trained on a GPU-enabled system. However, in a Cloud-based development environment, such a proposition can prove expensive as the GPU is under-utilized at 0% load during data ingestion and preprocessing stages of model development. This does not just increases the project development time, but it also elevates project cost. Therefore, there is an need for a preprocessing framework that can solve this problem.

In order to optimize GPU utilization and reduce total development time, this paper proposes a preprocessing framework for big scholarly data management, called `Preprocessing Pipeline for Scholarly Applications` or `P3SAPP`, which focuses on the creation of a data pipeline. This framework uses Spark ML for implementing APIs and parallelizing the different stages of data preparation in scholarly applications, which can greatly improve programmer productivity and reduce project cost. The running of the proposed framework also has high efficiency, in exchange for a minimal loss of accuracy.

The rest of the paper is organized in the following manner: Section 2 reviews related literature while Section 3 introduces the proposed framework alongside the baseline framework that shall be used for theoretical and experimental comparison. Section 4 illustrates the methodology. Details regarding implementation and evaluation of the proposed framework are provided in Section 5. The results obtained are analyzed and discussed in Section 6. Finally, conclusions and future work are synopsized in Section 7.

2 Related work

Big scholarly data consists of documents, typically in PDF or HTML format, with a structure consisting of several sections, including abstract, keywords, body and references. The constituents of these sections are essentially unstructured and consist of text, images and tables. Different applications make use of data from different or all sections. For instance, to create a citation graph [36], the text in the reference section needs to be scanned. In addition, scholarly data is also generated in the form of user logs. This data can be used for demographic analysis, system statistics and markers related to users and usage. Pig and Hive have been used for such analyses in existing literature [37].

Khan et al. [15] divide big scholarly data applications into five categories based on functionality. These categories include collaborator discovery, research management, expert finding systems, user logs analysis and other recommender systems. All these applications require data to be extracted from PDFs or webpages and bifurcated into sections from which relevant text is chosen for analysis. For example, collaborator discovery and expert finding depend primarily on author information and references. The rest of the textual information can be ignored for such applications.

Big scholarly data life cycle can be divided into six phases. The first step is the acquisition of scholarly documents in the form of PDFs or web pages [27]. During the second phase, data from these sources are extracted and collated into JSON or XML files. This process is referred to as 'extraction' [34]. Specific applications require specific textual information. Moreover, the variations in format and data specified, in a scholarly article, cause repeated and variable occurrence of nulls in the extracted data. Besides, multiple copies and versions of scholarly articles are available on the Internet. As a result, the presence of duplicates is highly probable. These preprocessing steps are essential for improving quality and value of scholarly big data regardless of the application.

Different applications have different preprocessing requirements. For instance, removal of duplicates while handling author information is more complicated than removing duplicates for numeric or textual entries such as DOI and titles. Author disambiguation is an outstanding preprocessing challenge as the currently available method manifests reasonable accuracy [17]. For most research management applications that focus on research article writing, concept development and references management, preprocessing tasks focus on cleaning text from title, abstract, paper body and/or references [18]. At the end of preprocessing, data is ready to be used as input for the model.

Depending on the application, different modeling techniques can be used for design and development of scholarly applications. Due to the textual nature of data, most applications use TF-IDF [5] or PageRank [11] for feature extraction. Common use cases include automatic keyword extraction [10] and topic modeling [39]. Recent applications have adopted machine and deep learning techniques for better accuracy.

The coverage of this work is limited to deep learning-based applications and the approaches to their implementation. During review, three recent scholarly applications that make use of deep learning were studied. Table 1 provides details about these applications, including implemented functionality and used technologies. All three applications use deep learning techniques on scholarly data. The implementation details of these applications indicate the use of conventional or baseline approach [1, 9, 21]. Review [40] suggests that Spark has not been used in any capacity for deep learning applications, particularly in the big scholarly data domain.

Literature review also suggests that the five methods, which are most commonly used and required for textual preprocessing include removal of punctuation, short words, stopwords, HTML tags and special characters, in addition to others. Finally, the results generated by the model are summarized and presented in the form of textual data or WordCloud [18] for better visualization. This work focuses on preprocessing techniques for deep learning-based scholarly applications. Therefore, it will not discuss modeling and visualization in detail.

Table 1 Existing deep learning application of big scholarly data

Application	Features	Preprocessing	Modeling Technique
1. Deep Key-phrase Generation [21]	Extracts key-phrases automatically using deep learning techniques	Lowercasing, tokenization and replacing digits with their string versions	RNN and Copy-RNN
2. Keyword extraction from scholarly documents using Bi-LSTM-CRF [1]	Solves the key-phrase extraction problem by modelling it as a sequence-labelling problem.	Abstract/Key-phrase data pairs are tokenized.	LSTM-CRF, CRF, Bi-LSTM, and LSTM
3. PubMender: A system for biomedical venue recommendation [9]	It is a journal recommendation system that works specifically in the biomedical domain.	NLTK for word segmentation	Venue recommendation is considered a multi-label classification problem and CNN is used.

Q2

The baseline approach uses a Pandas dataframe to ingest and store data, which is sequentially cleaned by performing tokenization, conversion to lowercase, and removal of HTML tags, unwanted characters, stopwords and short words, for each input entry. The output of this approach is a Pandas dataframe that has all the input elements in their cleaned form. Reference applications [1, 9, 21] make use of the baseline approach (`CA`). In order to facilitate comparison of the proposed approach (`P3SAPP`) with the baseline approach (`CA`), Table 2, provides the algorithm of the same.

Table 2 Algorithm for CA

Input: Data files
Output: Pandas dataframe with extracted and cleaned text
BEGIN
1. Initialize a Pandas dataframe, data.
2. *For each directory*
3. *For each file*
4. Read file into a dataframe
5. Select data to be extracted
6. Append the Pandas dataframe data with selected data
7. *END For*
8. *END For*
9. Remove NULL valued rows
10. Remove duplicates
11. *For all rows in the dataframe*
12. Perform text cleaning
13. *END For*
14. Remove NULL valued rows
END

Steps 2-8 perform ingestion and time corresponding to their execution is considered as ingestion time. Steps 9-10 perform pre-cleaning and time corresponding to their execution is considered as pre-cleaning time. Steps 11-13 perform cleaning and time corresponding to their execution is considered as cleaning time for CA. Step 14 performs post-cleaning. The post-cleaning times are correspondingly determined. The total preprocessing time for CA is determined by the execution time of steps 2-14. Theoretically, the algorithmic time complexity for conventional approach is `O(n)` because every element to the concerned column will be accessed and processed. It is noteworthy that `n` is the total number of elements in the column concerned.

3 Proposed framework

Preprocessing is one of the most crucial stages of the data lifecycle, which needs to be accurate as well as cost and time effective for development of sustainable applications. This section provides an overview of the preprocessing approach and describes the methodology used by the proposed approach to solve the problem of an excessively time-consuming preprocessing phase in deep learning – based scholarly applications.

Data preprocessing for deep learning–based scholarly applications involves three stages. Firstly, data from data files needs to be ingested into a dataframe. This dataframe is then prepared for text cleaning by filtering out unwanted elements such as null values and duplicates. This stage is referred to as pre-cleaning. Pre-cleaned data is fed to the cleaning stage, which outputs individually processed elements in the form of a dataframe. This dataframe is then fed to the post-cleaning stage for finalization of the preprocessing results. This stage again identifies nulls and removes them. This step is required because cleaning stage may also have introduced nulls, which need to be tackled. This work proposes a black-box approach that takes data files as input and provides Pandas dataframe as output. Therefore, if the cleaning stage provides output in any other dataframe format, its conversion to Pandas dataframe shall be performed in the post-cleaning stage.

The data lifecycle for any deep learning-based application includes four phases namely, data ingestion, data preprocessing, model training and model inference. The proposed framework identifies parallelizable phases of scholarly data lifecycle. These phases are fine-grained to create a data pipeline, which is escalated to Spark for reducing the total execution time of preprocessing. This research paper focuses on the preprocessing stage, which can be further divided into three sub-phases namely, pre-cleaning, cleaning and post-cleaning.

The proposed framework identified cleaning as a potentially parallelizable phase and escalated the same to Spark. The pre-cleaning phase needs to scan through the dataframe to identify duplicates and nulls for removal. Therefore, a sequential operation can be seen as uncomplicated and beneficial. On the other hand, the post-cleaning stage involves removal of nulls and conversion of the dataframe to a standard Pandas dataframe. The conversion part of this operation takes much more time than null removal. However, the dataframe format conversion cannot be parallelized because this framework directly uses the library function available for this purpose. The algorithm for the proposed approach is provided in Table 3.

The use of a distributed technology like Spark reduces the data ingestion time in view of the fact that data files are split and read. On the contrary, standard technology reads files sequentially for creation of dataframe. As a result, major benefits can be reaped in the data ingestion phase of the scholarly data lifecycle as well. The model training and inference stages of the scholarly data lifecycle remain untouched and have not been altered

Table 3 Algorithm for P3SAPP

Input: Data files
Output: Pandas dataframe with extracted
and cleaned text
BEGIN
1. Initialize a Spark dataframe, data.
2. *For each directory*
3.   *For each file*
4.   Read file data into a dataframe
5.   Select data to be extracted
6.   Perform union between Spark
            dataframe data and selected data
7.   *END For*
8. *END For*
9. Remove NULL valued rows
10. Remove duplicates
11. Define different stages of preprocessing APIs
12. Initialize Spark ML Pipeline for
preprocessing
13. Fit the data on Pipeline
14. Transform data using Pipeline
15. Convert Spark dataframe to Pandas dataframe
16. Remove NULL valued rowsEND
END

as part of this framework. It is noteworthy that the model training and inference stages are dependent on the application being developed. Therefore, application-specific model escalation to Spark for reduction in model development time can be attempted.

4 Methodology

The proposed framework uses a big data technology, Spark [40], for ingesting data and parallelizing specific phases of preprocessing stage, reducing the preprocessing time, which in turn reduces the total execution time. This reduction has a direct impact on the total time for which a Cloud-based GPU instance shall be required, correspondingly reducing the development time, computing cost and the overall project cost.

The overall framework can be broken down into four stages, out of which `P3SAPP` alters data ingestion and preprocessing stages. Steps 2-8 perform ingestion and time corresponding to their execution is considered as ingestion time. Steps 9-10 perform pre-cleaning and time corresponding to their execution is considered as pre-cleaning time. Step 14 performs cleaning for `P3SAPP` and its execution time corresponds for `P3SAPP`'s cleaning time. Steps 15-16 perform the same for `P3SAPP`. The post-cleaning times are correspondingly determined. Execution time for steps 2-16 is used to determine preprocessing time for `P3SAPP`. Theoretically, the algorithmic time complexity for the `P3SAPP` approach is `O(n/k)` where `k` is the number of nodes in the cluster if Spark is operating on cluster mode or the number of cores used to parallelize the job, if Spark is operating on local [*] mode. Typically, in local mode mode, the driver runs locally. However, in cluster mode, the driver runs on one of the worker nodes, which form the cluster.

4.1 Data ingestion

Data ingestion is the first stage of model development for any machine learning or deep learning application. As part of this stage, data is ingested into the system for further processing. Irrespective of the format of base dataset, this approach proposes ingestion of data into a PySpark dataframe [22]. Since the raw data from a scholarly document is structured in the sense that it can be ingested in the form of rows and columns, Spark SQL [23] has been selected as the base technology for operating with data inside Spark. Spark SQL provides a dataframe interface, which is capable of operating on different data formats, including JSON, ORC, Parquet and others [2]. Besides, Spark also provides generic data loading and saving methods, in which developers can specify their own working formats. This allows the flexibility to work with different formats using the same base technology. As a result, this framework can be used for generic purposes. The advantage of using a Spark dataframe is that relational transformation can be performed on data along with the provision to register a dataframe as a temporary view. Ingestion of data into a Spark dataframe is more efficient than the ingestion in Pandas [4].

4.2 Data preprocessing

For scholarly applications, the ingested data values are typically textual in nature. This text needs to be cleaned before it can be sent for further processing. On the basis of literature review, it has been deduced that commonly required text cleaning tasks include: (1) tokenize text, (2) convert text to lower case, (3) remove HTML tags, (4) remove unwanted characters, (5) remove stopwords and (6) remove short words. The Spark ML Feature package provides some APIs that are built on top of dataframes for feature transformation. For text preprocessing, the available APIs includes `Tokenizer`[1], for tokenizing text and `StopWordsRemover`[2], for removing stopwords. However, the rest of the APIs are not present and have been implemented in this work.

It is proposed that the APIs must be used to create a Spark ML Pipeline [24] so that Spark can perform the pipelined tasks in a parallel fashion, to reduce the time required. Typically, Spark ML Pipeline consists of transformers and estimators. `P3SAPP` proposes to use Pipelines for chaining multiple transformer APIs to specify a preprocessing workflow. On the basis of the preprocessing requirements, different transformer APIs can be chosen and chained in the pipeline for faster preprocessing. Finally, the resulting Spark dataframe is transformed into a Pandas dataframe, which can be fed to the model training sub-system. This sub-step is in line with the black box model [25]. The proposed approach does the same as the conventional approach, which takes raw data as the input, then generates Pandas frame as the output for subsequent model development. It is important to mention that future work intends to escalate model training and inference to Spark as well. Therefore, Spark ML is used so that this framework can be improved as it is for future work.

– `ConvertToLower`

[1] https://spark.apache.org/docs/latest/api/scala/org/apache/spark/ml/feature/Tokenizer.html
[2] https://spark.apache.org/docs/latest/ml-features.html#stopwordsremover

This API performs case conversion of all the row entries for the column provided as input. The case of all the alphabets in the entries is changed to lowercase. This API is essential in view of the fact that most NLP tasks require matching, similarity identification or manipulation based on identification of alphabets, words or strings. The use of such an API reduces programming effort by bringing all the values on the same level of casing.

– `RemoveHTMLTags`

Considering that the primary source of all scholarly data is the web and in most scenarios, it is required to ingest data using a crawler, textual data is typically retrieved as HTML content with tags. Although, this may or may not be true for all entries, it can be taken as a mandatory text-cleaning step before any analytical task can be performed.

– `RemoveUnwantedCharacters`

Once all the text is in lowercase and devoid of all tags, string-based manipulations can be performed. Common cleaning tasks require removal of the following characters or textual elements:

  – Punctuation
  – Text between parentheses
  – Apostrophes
  – Numbers and any special characters
  – Perform contraction mapping

This API performs textual cleaning by removing all the above-mentioned textual elements and outputs strings that have relevant words and phrases for advanced processing.

– `RemoveShortWords`

Some words such as abbreviations or conjunctions that are not typically removed using other APIs can be identified and removed on the basis of their word length. Therefore, this API cleans the text to ensure that smaller words such as abbreviations or variable names, which are comparatively insignificant information, can be removed. As part of this API, the user is expected to provide another input named threshold, which determines the maximum number of characters that a word should have for it to be considered for removal. Therefore, this API removes all words that are equal to or less than the threshold value in length.

4.3 Model training and inference

The model shall be developed on the basis of the required application and trained using the generated Pandas dataframe. The trained model can then be inferred to deduce the required results.

5 Evaluation

In order to test the feasibility of the proposed approach (`P3SAPP` and quantify its benefits in terms of time and cost, title or summary generation is chosen as the use case. The primary reason for choosing this use case is that it requires multi-level textual preprocessing. Moreover, title or summary generation from abstracts for scholarly articles is an application that can be used in many different ways, which include article review management system that can generate summary of received articles to facilitate editorial decision on a manuscript.

Moreover, research article writing applications that can automatically suggest appropriate titles for a scholarly article on the basis of provided abstract, can also be a target application.

5.1 Ingestion phase

Data, which is available in the form of JSON files, is ingested into Spark dataframe using API provided for the same. It is important to note that only data corresponding to titles and abstracts is ingested.

5.2 Preprocessing phase

The preprocessing stage is divided into three sub-stages namely, pre-cleaning, cleaning and post-cleaning. For the conventional approach, the three stages perform the following functions:

- The pre-cleaning stage removes nulls and duplicates.
- The cleaning stage performs different set of operations on titles and abstracts. For abstracts, text is converted to lowercase and HTML tags, unwanted characters, stopwords and short words are removed. On the other hand, for abstracts, text is converted to lowercase and HTML tags, unwanted characters and short words are removed. The implemented APIs - `ConvertToLower`, `RemoveHTMLTags`, `RemoveUnwantedCharacters` and `RemoveShortWords` were used. Although, `StopWordsRemover` is a generic API available for stopwords removal, the use case - specific implementation for the same was also done.
- The cleaning stage may introduce nulls. Therefore, the post cleaning stage again checks for any nulls and removes them.

At the end of the post-cleaning stage, the Pandas dataframe is ready to be imported into the model training module. The proposed approach performs the same set of steps for the three different stages of preprocessing. However, all the transformational operations are performed on Spark dataframe and it is converted to a Pandas dataframe during the post-cleaning stage. The preprocessing workflows required for abstracts and titles are different and shown in Figs. 1 and 2. Since, the abstract will be used as feature for training the model, it must be completely clean. Therefore, the cleaning tasks performed for abstracts include:

- Convert all the text to lowercase.
- Remove all HTML tags if any.
- Remove all unwanted characters.

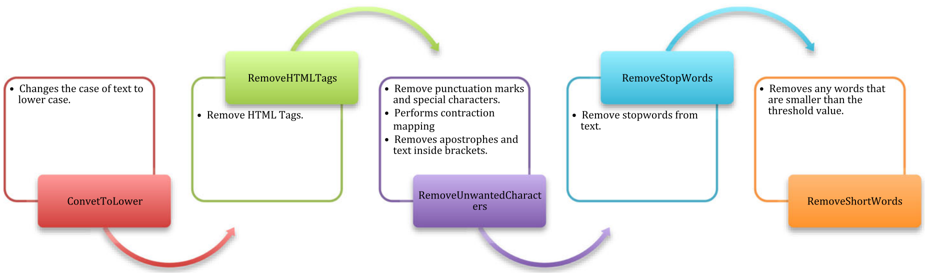


Fig. 1 Preprocessing pipeline for cleaning abstracts

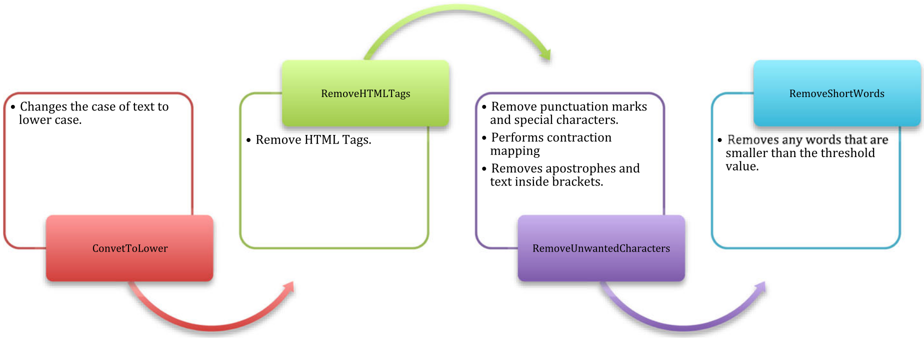


Fig. 2 Preprocessing pipeline for cleaning titles

- Remove stopwords.
- Remove short words.

On the other hand, title is the target for the model and thus, the cleaning tasks required include:

- Convert all the text to lowercase.
- Remove all HTML tags if any.
- Remove all unwanted characters.
- Remove short words.

For the purpose of implementing the chosen case study, the threshold value for short words removal is fixed at threshold = 1. This will remove words that are 1-character in length, keeping all other words to ensure maximum information is used for summary generation. This value can be increased depending upon the use case. The respective APIs are called to define Spark ML pipelines. The pipelines are fitted to data and the input dataframe is transformed using this pipeline.

5.3 Model training and inference phase

The chosen case study implements text summarization for scholarly articles. There are two types of text summarization methods namely, abstractive text summarization [34] and extractive text summarization [35]. Extractive text summarization identifies and extracts sentences, phrases and words from the original text, while abstractive text generates new sentences that summarize the original text. The problem of title generation from abstract of scholarly article requires abstractive text summarization.

Text is sequential information and requires seq2seq modeling [36] where the input sequence is a long text while the output sequence is its summary or short text. Therefore, generating title from abstract of a scholarly article is a many-to-many seq2seq problem. The seq2seq model is composed of two components namely, encoder and decoder. These components are implemented using variants of Recurrent Neural Networks (RNN) [37] such as Long Short Term Memory (LSTM) [38] or Gated Recurrent Neural Network (GRU) [39]. The reason for this assertion is that RNNs are better capable of handling the vanishing gradient problem. As a result, they can capture long-term dependencies more efficiently. The setting up of the encoder and decoder is divided into two phases namely training phase and inference phase. The details of implementation are as follows:

– Training

In the training phase, the encoder and decoder are set up. The model is trained to make a prediction of the target sequence offset per time-step. Therefore, at each time-step, the encoder LSTM processes the input sequence to feed one word into the encoder. The encoder's job is to comprehend and learn the input sequence's contextual information. The encoder architecture is illustrated in Fig. 3. It is important to note that h_i and c_i are hidden and cell states respectively. Since, encoder and decoder are different stages, the hidden and cell states are fed to the decoder for initialization.

It is the decoder's job to read target sequence and make predictions on the basis of sequence offset per time-step. Therefore, every next word is predicted using the previous word. The decoder architecture is illustrated in Fig. 4. Since the target sequence's first word is unknown, the first word passed to the decoder is $< start >$ token and the $< end >$ token marks the end of sentence.

In order to build a model, a 3-layer stacked LSTM is used for encoder. Using a stacked LSTM ensures better sequence representation. The model is instructed to stop early when the validation loss begins to increase. This is performed to optimize the number of epochs executed for model building.

– Inference

The encoder and decoder of LSTM are setup for the inference stage. Figure 5 illustrates the model inference architecture. The steps for model inference are provided in Table 4.

There are certain limitations of this training architecture. The job of an encoder is to convert the complete input sequence into a vector of fixed length. This approach works well for short sequences. However, when dealing with long sequences, the model may suffer from inability to memorize the input sequence into a fixed length vector. In order to solve this problem, attention mechanism [40], which modifies the approach in the sense that the model is now attentive to important sub-sequences in the input focusing on the whole input sequence.

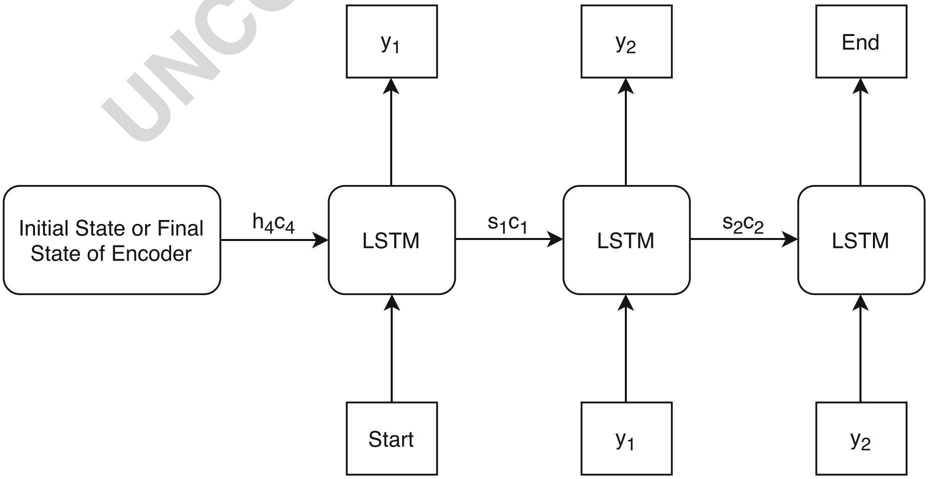


Fig. 3 Training phase: LSTM encoder architecture

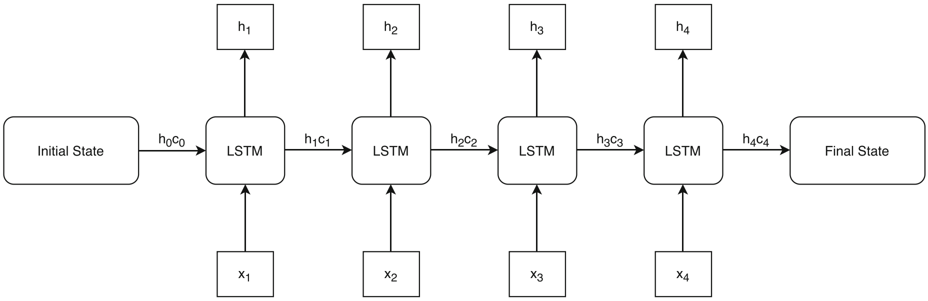


Fig. 4 Training phase: LSTM decoder architecture

As seen in Figs. 3 and 4, the encoder generates a hidden state h_j for every j time-step and decoder generates the hidden state s_i for every i time-step. The alignment of the source word (alignment score or e_{ij}) with the target word is calculated using the score function, which is given by (1):

$$e_{ij} = score(s_i, h_i) \tag{1}$$

There are many types of score function such as dot product, additive and generic score function. Once the alignment score is calculated, the softmax function is used for normalizing the scores and getting attention weights. (2) describes the mathematical computation for attention weights (a_{ij}).

$$a_{ij} = e^{e_{ij}} \sum_{k=1}^{T_x} e^{e_{ik}} \tag{2}$$

A linear sum of products is computed with attention weights and encoder's hidden states to determine attended context vector (C_i), which is given by (3).

$$C_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{3}$$

The attended hidden vector (S_i) is computed by concatenating the attended context vector and decoder's target hidden state for time-step i and is given by (4).

$$S_i = concatenate([S_i, C_i]) \tag{4}$$

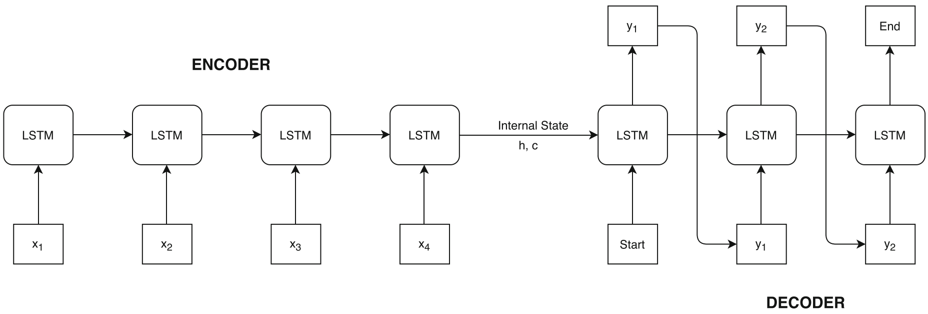


Fig. 5 Inference phase architecture

Table 4 Algorithm for inference phase

Input: $< start >$ token

Output: Generated text string

BEGIN

1. The entire input sequence is encoded. The generated internal states are fed to the decoder for initialization.
2. The $< start >$ token is given as input.
3. The decoder is run for one time-step.
4. The next word is determined with probability of occurrence. The word with the highest probability is chosen.
5. The generated word is passed as input to the decoder for next time-step. The internal states are also updated according to the time-step.
6. Steps 3-5 are repeated until maximum limit of word generation is reached or $< end >$ is generated.

END

The attended hidden vector S_i is given to the dense layer for computation of y_i, which is given by (5).

$$y_i = dense(S_i) \tag{5}$$

The implementation of text summarization model is inspired by Pai's Keras implementation [41] for text summarization. Since Keras does not have an inbuilt attention mechanism, Ganegedara's implementation [42] of Bahdanau attention mechanism [40] has been used for the case study.

5.4 Experimental setup

The development and testing environment makes use of a GPU of the following configuration: Tesla K80 – 12 GB Memory and 61 GB RAM – 100 GB SSD. The CPU configuration used is Intel Xeon with 2 cores, 8 GB Memory and 200 GB SSD. FloydHub was used to provision the requires resources from a Cloud-based environment. Spark version v2.4.4 in local [*] mode was used for all experimentation purposes.

5.5 Dataset

In order to implement a deep learning model for text summarization, a dataset with titles and abstracts was chosen. For this contribution, the CORE[3] dataset with the schema shown below was selected because it is open access. The full dataset is a zipped file of 330 GB size. The unzipped version expands to 1.44 TB. It includes 123M metadata items with 85.6M items containing abstracts.

The complete dataset includes 2085 JSON files of variable size. For the purpose of this research, five subsets were created. The sizes of the datasets used for the five use cases are 4.18 GB, 8.54 GB, 13.34 GB, 18.23 GB and 23.58 GB. The files are selected in such a manner that datasets are composed of different number of files, with each file variably sized, ranging from sizes of the order of KB to GB. Moreover, an incremental approach is used

[3] https://core.ac.uk/services/dataset/

for increasing the dataset size because a completely changed dataset may induce a changed behaviour from the system.

5.6 Results

The testing and evaluation of the `P3SAPP` intends to capture the variations in execution time and accuracy for both the approaches. Finally, the obtained results are used to estimate the impact of `P3SAPP` on the cost of the project.

5.6.1 Execution time

The total time required for execution of a deep learning application is given by:

$$T = t_i + t_{pp} + (n * t_{mt}) + t_{mi} \tag{6}$$

The variables used in (6) are as follows:

T = Total execution time
t_i = Data ingestion time
t_{pp} = Preprocessing time
n = Number of epochs
t_{mt} = Model training time
t_{mi} = Model inference time

In all experiments, value of t_{mi} for generating a single summary was approximately the same, with the following value: $t_{mi} \sim 2 seconds$. Therefore, the value of t_{mi} is negligible in comparison to t_i, t_{pp} and t_{mt}. It is for this reason that the value of t_{mi} is ignored for total time computation and cost analysis. Besides this, cumulative time (t_c) is given by:

$$t_c = t_i + t_{pp} \tag{7}$$

Thus, the revised equation is follows:

$$T = t_c + (n * t_{mt}) \tag{8}$$

The proposed approach reduces cumulative time (t_c); the results for which are provided in the sections given below.

- Ingestion Time

  Ingestion time is defined as the time to ingest data from multiple JSON files into a Spark dataframe. The values of ingestion time determined in the performed experiments are given in Table 5. The results indicate a consistent reduction in ingestion time for variable dataset sizes. These results can consequently be inferred from the graphical illustration of the results. Figure 6 illustrates ingestion time variations with respect to dataset size. While the conventional approach shows staggering growth with ingestion time shooting up for higher dataset sizes, P3SAPP manifests a slower increase in ingestion time with increase in dataset size. Moreover,ingestion time is reduced by more than 99% for datasets larger than 5 GB.
- Preprocessing time

  Preprocessing time is the total time required by the system to clean ingested data. The total preprocessing time is derived from (4) and computed using (9). The values of pre-cleaning, cleaning, post-cleaning and total preprocessing time determined in the performed experiments are given in Table 6. Fig. 7 illustrates the trends for

Table 5 Comparison of Ingestion Time for CA and P3SAPP

Dataset ID	Dataset Size	Ingestion Time		
	(GB)	CA	P3SAPP	Reduction (%)
1	4.18	433.631	13.076	**96.984**
2	8.54	3542.393	26.253	**99.259**
3	13.34	8701.101	79.843	**99.082**
4	18.23	17139.434	93.637	**99.454**
5	23.58	32698.916	104.055	**99.682**

Q3

preprocessing times obtained for conventional and proposed approaches.

$$t_{pp} = t_{prc} + t_c + t_{poc} \tag{9}$$

The variables used in (9) are as follows:

t_{pp} = Total preprocessing time
t_{prc} = Pre-cleaning time
t_c = Cleaning time
t_{poc} = Post- cleaning time

The rise in preprocessing time for conventional approach is steeper than the same obtained for the proposed approach, exhibiting an average reduction of approximately 40%. It is important to note that cleaning stage takes a large amount of time for conventional approach. On the other hand, conversion of Spark dataframe to Pandas dataframe in the post-cleaning stage consumes most of the total preprocessing time for the proposed approach.

– Cumulative time

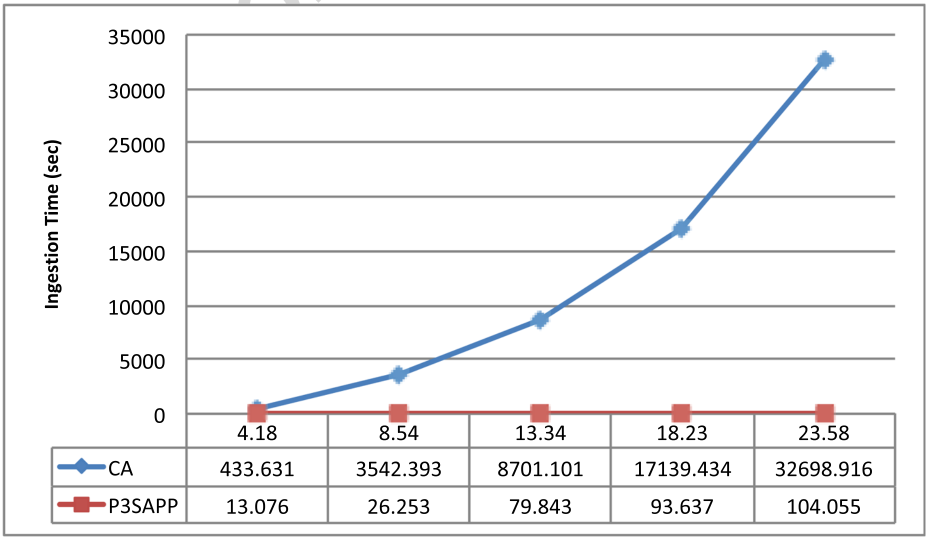


Fig. 6 Comparative analysis of ingestion time

Table 6 Comparison of preprocessing time for ca and P3SAPP

Dataset ID	Dataset Size	Pre-Cleaning (in seconds)		Cleaning (in seconds)		Post-Cleaning (in seconds)		Total Preprocessing Time (in seconds)		
	(GB)	CA	P3SAPP	CA	P3SAPP	CA	P3SAPP	CA	P3SAPP	Reduction (%)
1	4.18	0.165	0.009	154.394	0.161	0.118	89.31	154.679	89.485	**42.148**
2	8.54	0.273	0.008	232.223	0.154	0.247	140.442	232.745	140.609	**39.589**
3	13.34	0.528	0.008	457.768	0.172	0.452	262.307	458.94	262.492	**42.8**
4	18.23	0.811	0.017	628.464	0.206	0.635	351.62	629.913	351.848	**44.143**
5	23.58	1.067	0.017	862.453	0.252	0.887	477.51	864.409	477.784	**44.727**

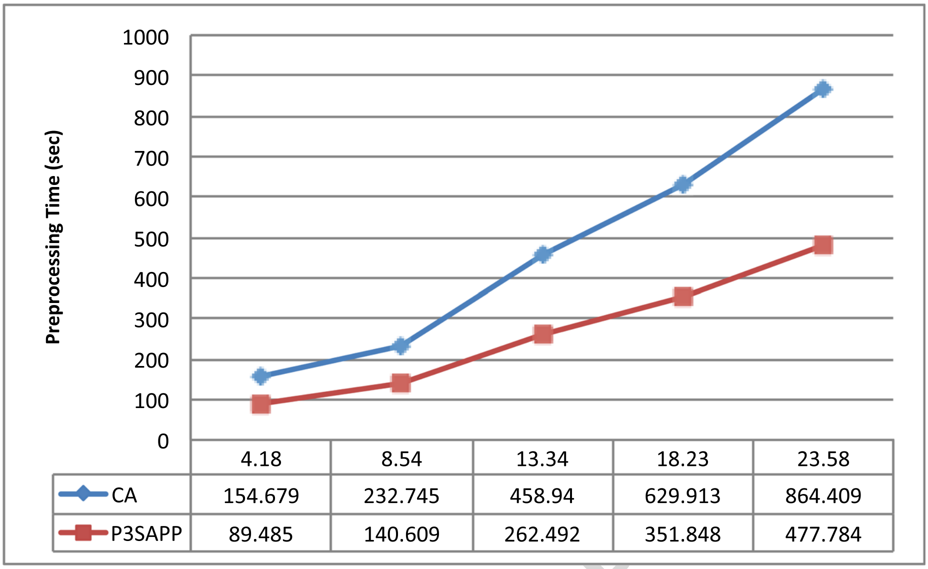


Fig. 7 Comparative analysis of preprocessing time

Cumulative time is sum of ingestion and preprocessing times and is calculated using (2). The trend for cumulative time obtained using conventional approach exhibits staggering growth while the proposed approach manifests a very slow escalation. The reduction in cumulative time is increasing with increase in dataset size, making this approach more beneficial for larger datasets. The values of cumulative time determined in the performed experiments are given in Table 7. Figure 8 illustrates variations in cumulative time with rise in dataset size.

5.6.2 Accuracy

The accuracy for the proposed approach, P3SAPP, is determined by the percentage of matching records in the Pandas dataframes generated for conventional (`CA`) and proposed approaches (`P3SAPP`). The extracted records in the form of a Pandas dataframe for both the approaches were compared to determine the matching records and consequently, the percentage of matching records. The results obtained for accuracy are provided in Tables 8 and

Table 7 Comparison of cumulative time for CA and P3SAPP

Dataset	ID Dataset Size	Total Time		
	(GB)	CA	P3SAPP	Reduction (%)
1	4.18	588.31	102.561	**82.567**
2	8.54	3775.138	166.862	**95.58**
3	13.34	9160.041	342.335	**96.263**
4	18.23	17769.347	445.485	**97.493**
5	23.58	33563.325	581.839	**98.266**

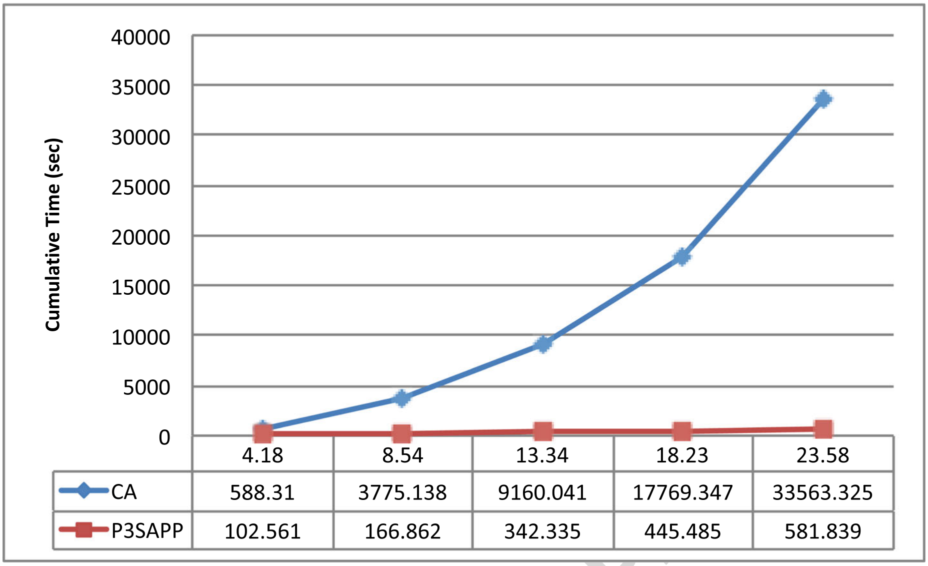


Fig. 8 Comparative analysis of cumulative time

9. The average accuracy for titles was determined to be 96.595%. On the other hand, the average accuracy for abstracts was found to be 97.929%.

5.6.3 Cost benefit analysis

Cloud-based services like AWS[4], GCP[5] and FloydHub[6] provision Platform-as-a-Service (PaaS), on hourly expenditure. Therefore, the total cost can be estimated on the basis of the number of hours a job will take to complete. The total time for conventional and proposed approach can be computed using (8). For cost benefit evaluation, the number of epochs is fixed as 10, 25 and 50.

Cost benefit is determined by converting total time in hours and multiplying the value with hourly cost. The formula for cost evaluation is given by,

$$C = x * T \tag{10}$$

In (10), C is the total cost of execution and x is hourly cost. Using (10), cost benefit is given by,

$$CB = \frac{x * (T_{ca} - T_{pa})}{x * T_{ca}} * 100 \tag{11}$$

$$CB = \frac{T_{ca} - T_{pa}}{T_{ca}} * 100 \tag{12}$$

In (12), CB is Cost Benefit, (T_{ca}) is total time taken for conventional approach (CA) and (T_{pa}) is total time taken for proposed approach. The results of the computation performed for determination of T and CB are provided in Table 10. Results indicate an escalation in

[4]https://aws.amazon.com/emr/features/spark/
[5]https://Cloud.google.com/dataproc/
[6]https://www.floydhub.com/product/build

Table 8 Accuracy for titles

Dataset ID	Conventional Approach (CA)	Proposed Approach (P3SAPP)	Matching Records	Percentage (%)
1	88709	88709	86935	98
2	132683	132683	128924	97.167
3	256362	256362	248950	97.109
4	345169	345169	334881	97.019
5	480712	480712	450333	93.68

cost benefit with increase in dataset size. However, as the number of epochs increase, the corresponding cost benefit is lowered, as is evident from Fig. 9.

It can be deduced from the results provided in Table 10 and graphical illustration shown in Fig. 9, that cost is minimal for larger datasets and higher epochs. This is relevant with regard to the scalability requirement of big data systems. As dataset size and number of epochs chosen for model development increase, optimum cost benefit can be expected
Q4 (Figs. 10 and 11).

6 Discussion

Evidently, the ratio of time saving and MTT/epoch increases exponentially with escalation in dataset size (as shown in Fig. 12). Moreover, for Dataset ID = 5, this value is as high as 7.9, which means the time savings provided by the proposed approach is equal to the time taken by 7.9 epochs. The significance of this value can translate into major time and cost savings for projects that work with larger datasets. Results indicate that cost benefit is expected to escalate with increase in dataset size for a fixed number of epochs. Although, the proposed approach records high accuracy in terms of matching records produced by the two approaches, it is noteworthy that accuracy reduces for larger datasets, but remains more than 93%. The reason for non-matches between records from the two dataframes can be attributed to the difference in ingestion methods. Reduction in this parameter and the impact in variations in matching records on the generated model shall be studied as future
Q5 work (Table 11).

Another important point to note is that the proposed approach has been implemented and tested with Spark on local [*] mode, which means that Spark is running locally and the different worker threads are working on the different logical cores on the machine. Spark

Table 9 Accuracy for abstracts

Dataset ID	Conventional Approach (CA)	Proposed Approach (P3SAPP)	Matching Records	Percentage (%)
1	88709	88709	88282	99.519
2	132683	132683	129179	97.359
3	256362	256362	251572	98.131
4	345169	345169	339541	98.369
5	480712	480712	462766	96.267

Table 10 Cost benefit analysis

Dataset ID	Cumulative Time (secs)		MTT per epoch	Total Time for 10 epochs (hrs)			Total Time for 25 epochs (hrs)			Total Time for 25 epochs (hrs)		
	CA	P3SAPP	(secs)	CA	P3SAPP	Cost Benefit (%)	CA	P3SAPP	Cost Benefit (%)	CA	P3SAPP	Cost Benefit (%)
1	588.31	102.561	1132	3.31	3.173	**4.079**	8.024	7.89	**1.681**	15.886	15.751	**0.849**
2	3775.138	166.862	1698	5.765	4.763	**17.385**	12.84	11.838	**7.805**	24.632	23.63	**4.049**
3	9160.041	342.335	3166	11.339	8.889	**21.601**	24.53	22.081	**9.985**	46.517	44.067	**5.265**
4	17769.347	445.485	4070	16.241	11.429	**29.629**	33.12	28.388	**14.495**	61.464	56.651	**7.829**
5	33563.325	581.839	4170	20.906	11.745	**43.821**	39.44	29.12	**26.166**	67.24	58.078	**13.625**

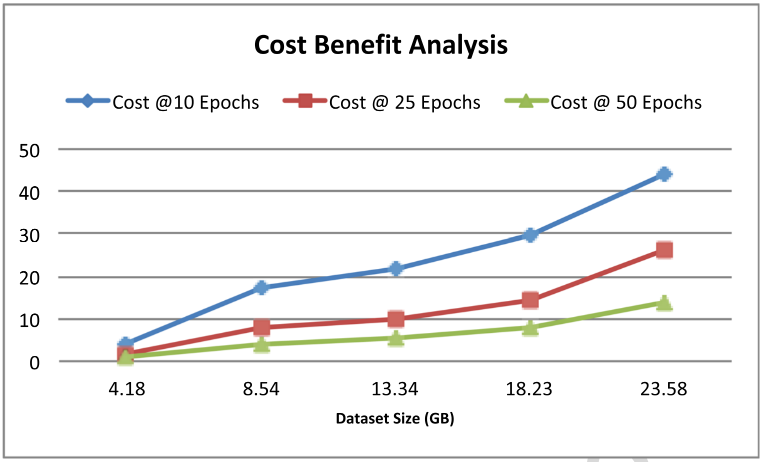
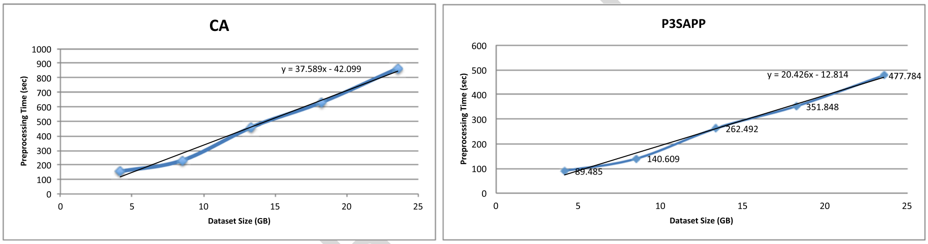


Fig. 9 Epoch-wise cost benefit comparison



(a) (b)

Fig. 10 Trend-line graphs for preprocessing results

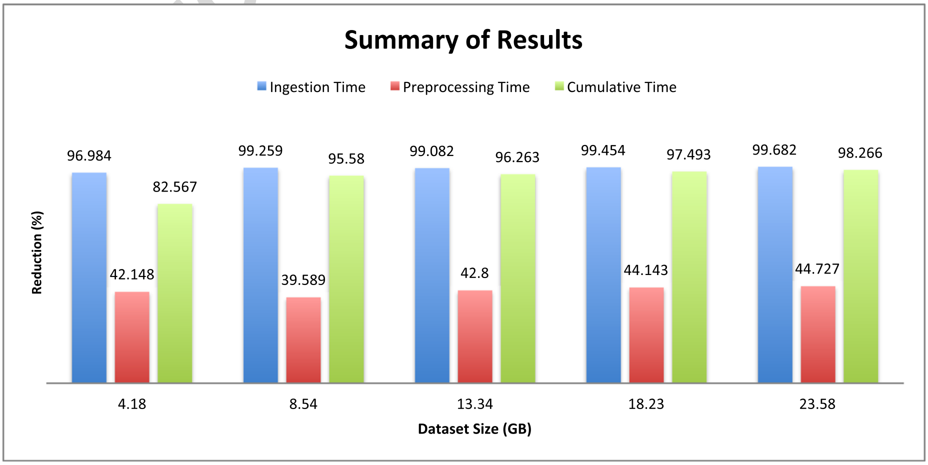


Fig. 11 Summary of results for execution time

Table 11 Reduction in preprocessing time in terms of MTT per epoch

Dataset ID	Dataset Size (GB)	Number of Training Records	Number of Validation Records	MTT per epoch (secs)	Time Saving (sec)	Ratio of Time Saving and MTT per epoch
1	4.18	70505	7834	1132	485.749	0.429
2	8.54	104368	11597	1698	3608.296	2.125
3	13.34	200908	22327	3166	9160.041	2.893
4	18.23	270514	30023	4070	17323.862	4.256
5	23.58	383002	42536	4170	32981.486	7.909

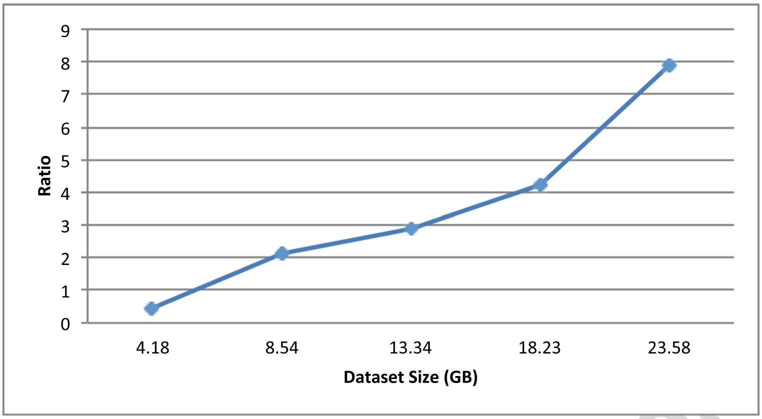


Fig. 12 Ratio of time saving and MTT/Epoch

can be moved to a full-fledged cluster to get enhanced results. Although, this work does not present any such results and shall be performed in the future. Moreover, escalating the deep learning model to Spark can also be explored in the future to reduce development time and costs, further. As it can be seen in Table 6, the variations in the total preprocessing time arises due the post cleaning time in the conversion of Spark dataframe to Pandas dataframe. Escalation of model to Spark shall remove this aspect of the proposed approach. Therefore, for a given configuration of Spark, the preprocessing time, in such a scenario, will be constant (Fig. 12).

Q6

7 Conclusion

This work proposes a framework that modifies the preprocessing stages of deep learning model development. Preprocessing, particularly for scholarly applications, is highly resource intensive. As a result, for application development that uses Cloud provisioned platforms and infrastructure, most of the time is wasted, as GPU remains underutilized during this time. Reducing the preprocessing time reduces the time of underutilization and overall cost of the project. The proposed approach provides more than 90% reduction in total preprocessing time, which includes ingestion and preprocessing time, for datasets larger than 5 GB. Besides this, the cost saving are dependent on the number of epochs and size of datasets. Cost savings are highest for lesser epochs and large datasets. It is important to note that both cost saving and reductions in cumulative time increase with increase in dataset size, making this approach highly relevant for big datasets. This shall also improve the accuracy of the developed deep learning model.

This work uses text summarization as a case study for framework evaluation. It may be tested for other NLP-based scholarly applications in the future to prove the generic validity of the framework. The accuracy of the approach in terms of matching records obtained when compared to conventional approach is more than 90% for datasets larger than 5 GB. The cause of mismatches is rooted in differences in ingestion. Further investigations to improve results for this aspect of the approach shall be attempted in the future. As part of this work, four APIs were implemented for enhancing the Spark ML feature class. More APIs can be identified and implemented in the future. The proposed model has used Spark on local

[*] model, which parallelizes different threads on different logical cores. Higher levels of parallelization can be investigated in future work. Moreover, escalation of the deep learning model to Spark will also be explored in the future.

References

1. Alzaidy R, Caragea C, Giles CL (2019) Bi-LSTM-CRF sequence labeling for keyphrase extraction from scholarly documents. In: In the world wide web conference, ACM, pp 2551–2557
2. Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Frankliny MJ, Ghodsi A, Zaharia M (2015) Spark SQL: Relational data processing in spark. In: Proceedings of the ACM SIGMOD international conference on management of data, pp 1383–1394. https://doi.org/10.1145/2723372.2742797
3. Beel J, Gipp B, Langer S, Breitinger C (2015) Research-paper recommender systems: a literature survey .Int J Digit Libr
4. Chen DY (2017) Pandas for everyone:Python data analysis. Addison-Wesley Professional
5. Chen J, Zhuge H (2019) Automatic generation of related work through summarizing citations. Concurr Comput, 31, 3. https://doi.org/10.1002/cpe.4261
6. Duari S, Bhatnagar V (2019) sCAKE: Semantic connectivity aware keyword extraction. Inf Sci (Ny) 477:100–117
7. Eisenstein J (2019) Introduction to natural language processing. MIT Press
8. Fang C, Mu D, Deng Z, Wu Z (2017) Word-sentence co-ranking for automatic extractive text summarization. Expert Syst Appl 72:189–195
9. Feng X, Zhang H, Ren Y, Shang P, Zhu Y, Liang Y, Xu D (2019) The deep Learning–Based recommender system “Pubmender” for choosing a biomedical publication venue: Development and validation study. J Med Internet Res 21(5):e12957 Q7
10. Florescu C, Caragea C (2017) Positionrank: an unsupervised approach to keyphrase extraction from scholarly documents. In: Proceedings of the 55th annual meeting of the association for computational linguistics, pp 1105–1115
11. Frank MR, Wang D, Cebrian M, Rahwan I (2019) The evolution of citation graphs in artificial intelligence research. Nat Mach Intell 1(2):79
12. Gandomi A, Haider M (2015) Beyond the hype big data concepts, methods, and analytics. Int J Inf Manage 35(2):137–144
13. Ganegedara T (2019) Keras layer implementation of Attention
14. Goyal P, Dollár P, Girshick R, Noordhuis P, Wesolowski L, Kyrola A, Tulloch A, Jia Y, Kaiming H (2017) Accurate, large minibatch SGD: Training ImageNet in 1 Hour, Retrieved from. arXiv:1706.02677
15. Khan S, Liu X, Shakil KA, Alam M (2017) A survey on scholarly data: From big data perspective. Inf. Process. Manag 53(4):923–944. https://doi.org/10.1016/j.ipm.2017.03.006
16. Khan S, Shakil KA, Alam M (2016) Educational intelligence: Applying cloud-based big data analytics to the indian education sector. Proc 2016 2nd Int Conf Contemp Comput Informatics, IC3I 2016 pp 29–34. https://doi.org/10.1109/IC3I.2016.7917930
17. Kim J, Diesner J, Kim H, Aleyasen A, Kim HM (2015) Why name ambiguity resolution matters for scholarly big data research, Proc -2014. IEEE Int. Conf. Big Data, IEEE Big Data 2014, pp 1–6. https://doi.org/10.1109/BigData.2014.7004345
18. Liu J, Tang T, Wang W, Xu B, Kong X, Xia F (2018) A survey of scholarly data visualization. IEEE Access 6, pp 19205–19221. https://doi.org/10.1109/ACCESS.2018.2815030
19. Liu P, Qiu X, Xuanjing H (2016) Recurrent neural network for text classification with multi-task learning. IJCAI Int Jt Conf Artif Intell 2016-Janua, pp 2873–2879
20. Maake BM, Ojo SO, Zuva T (2019) A survey on data mining techniques in research paper recommender systems. In: Research data access and management in modern libraries. IGI Global, pp 119–143
21. Meng R, Zhao S, Han S, He D, Brusilovsky P, Chi Y (2017) Deep keyphrase generation. ACL 201 - 55th, Annu Meet Assoc Comput Linguist Proc Conf, (Long Pap. 1), pp 582–592. https://doi.org/10.18653/v1/P17-1054
22. Mishra RK, Raman SR (2019) PySpark SQL recipes. Apress
23. Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of SPARQL. ACM Trans Database Syst 34(3):1–45

24. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Xin D (2016) Mllib: Machine learning in apache spark. J Mach Learn Res 17(1):1235–1241
25. Tiwana A (2004) Beyond the black box: knowledge overlaps in software outsourcing. Ieee Software 21(5):51–58
26. Nallapati R, Zhou B, Santos CD, Gulçehre Ç, Xiang B (2016) Abstractive text summarization using sequence-to-sequence RNNs and beyond. CoNLL 2016 - 20th SIGNLL Conf Comput Nat Lang Learn Proc, pp 280–290. https://doi.org/10.18653/v1/k16-1028
27. Ororbia AG, Wu J, Khabsa M, Williams K, Giles CL (2015) Big scholarly data in citeseerx: Information extraction from the web. WWW 2015 Companion - Proc. 24th, Int Conf World Wide Web, pp 597–602. https://doi.org/10.1145/2740908.2741736
28. Pai A (2019) How-to-build-own-text-summarizer-using-deep-learning. Retrieved from. https://github.com/aravindpai/How-to-build-own-text-summarizer-using-deep-learning/blob/master/How_to_build_own_text_summarizer_using_deep_learning.ipynb
29. Siddiqi S, Sharan A (2015) Keyword and keyphrase extraction techniques: A literature review. Int J Comput Appl 109(2):18–23. https://doi.org/10.5120/19161-0607
30. Tai KS, Socher R, Manning CD (2015) Improved semantic representations from tree-structured long short-Term memory networks. ACL-IJCNLP 2015 - 53rd Annu Meet Assoc Comput Linguist 7th Int Jt Conf Nat Lang Process Asian Fed Nat Lang Process Proc Conf 1:1556–1566. https://doi.org/10.3115/v1/p15-1150
31. Tang D, Qin B, Liu T (2015) Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of the 2015 conference on empirical methods in natural language processing, pp 1422–1432
32. Tanijiri J, Ohta M, Takasu A, Adachi J (2016) Important Word Organization for Support of Browsing Scholarly Papers Using Author keywords. In: Proceedings of the 2016 ACM Symposium on document engineering. ACM, pp 135–138
33. Tkaczyk D, Szostek P, Fedoryszak M, Dendek PJ, Bolikowski Ł (2015) CERMINE: automatic extraction of structured metadata from scientific literature. Int J Doc Anal Recognit 18(4):317–335
34. Tuarob S, Bhatia S, Mitra P, Giles CL (2013) Automatic detection of pseudocodes in scholarly documents using machine learning, Proc Int Conf Doc Anal Recognition, ICDAR, pp 738–742. https://doi.org/10.1109/ICDAR.2013.151
35. Wang D, Liang Y, Xu D, Feng X, Guan R (2018) A content-based recommender system for computer science publications. Knowledge-Based Syst 157:1–9
36. West JD, Wesley-Smith I, Bergstrom CT (2016) A recommendation system based on hierarchical clustering of an article-level citation network. IEEE Trans Big Data 2(2):113–123. https://doi.org/10.1109/tbdata.2016.2541167
37. Wu Z, Wu J, Khabsa M, Williams K, Chen HH, Huang W, Tuarob S, Choudhury SR, Ororbia A, Mitra P, Giles CL (2014) Towards building a scholarly big data platform: Challenges, lessons and opportunities. Proc ACM/IEEE Jt Conf Digit Libr, pp 117–126. https://doi.org/10.1109/JCDL.2014.6970157
38. Yang Z, Yang D, Dyer C, He X, Smola A, Hovy E (2016) Hierarchical attention networks for document classification. In: Proceedings of the 2016 conference of the north american chapter of the association for computational linguistic, pp 1480–1489
39. Yu D, Wang W, Zhang S, Zhang W, Liu R (2017) Hybrid self-optimized clustering model based on citation links and textual features to detect research topics. PLoS One 12(10):e0187164
40. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: Cluster computing with working sets. HotCloud 10 10:95 Q8
41. Zhang Q, Yang LT, Chen Z, Li P (2018) A survey on deep learning for big data. Inf Fusion 42:146–157
42. Zhou Y, Liu C, Yan P (2016) Modelling sentence pairs with tree-structured attentive encoder. COLING 2016 - 26th, Int Conf Comput Linguist Proc COLING 2016 Tech Pap, pp 2912–2922

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES:

Q1. City has been provided in affiliation please check if it is correct.
Q2. Please check all tables if captured and presented correctly.
Q3. Please provide significance for bold entries found in Tables 5, 6, 7 and 10. Otherwise, please remove emphasis.
Q4. Missing citation for Figures 10 and 11 were inserted here. Please check if appropriate. Otherwise, please provide citation for Figures 10 and 11. Note that the order of main citations of figures in the text must be sequential.
Q5. Citation for Table 11 was inserted here. Please check if appropriate. Otherwise, please provide citation for Table 11. Note that the order of main citations of tables in the text must be sequential.
Q6. Dummy citation for Figure 12 was inserted here. Please check if appropriate. Otherwise, please provide citation for Figure 12. Note that the order of main citations of figures in the text must be sequential.
Q7. Please check if the page range provided in reference 9 is correct.
Q8. Please check if the page range provided here is correct.