Towards API Documentation Information Extraction: Challenges And Approaches

by

Mehdi Assefi

(Under the Direction of Hamid R. Arabnia)

Abstract

The number and variety of web APIs are growing exponentially. Software engineers need to spend a significant amount of time and effort reading and understanding the requisite documentation. Moreover, this is not always a simple task since API documentation from a provider can be anything from a single HTML page to a complex elaboration of detailed information spanning several pages. Comprehending such a wide variety of API documentation styles is therefore a labor-intensive and error-prone task for engineers. By providing a machine-learning platform that can extract and standardize API usage information, we believe we can accelerate the creation of API-enabled systems by using automation to streamline comprehension. The study below introduces an approach to automate and standardize usage information for APIs by designing several novel, machine-learning algorithms in order to extract key attributes from API documentation and to generate a machine-readable Open API Specification (OAS).

We designed and implemented

- a Spark-based web-crawler to collect raw HTML pages and apply several deep-learning models to identify REST API documentation, known as Swagger. The collected documents that follow Swagger formatting will then be used as training data for the reminder of the components;
- a content-based learning model that identifies the context of a block of extracted API features;
- a novel, signature-based machine-learning model that recognizes a sequence of successful/unsuccessful extracted API endpoints; and
- a novel deep-mapping model that pinpoints fine-grained mapping of extracted API attributes to OAS objects.

The results of our experiments demonstrate that the proposed approach works successfully with an accuracy of 99%, 94% and 97% for content-based learning, signature-based learning, and Deep Mapping of API attributes respectively. We then use the models to produce OAS-compliant API specifications for more than 2,585 public APIs, validate them via API calls, and finally deploy the validated APIs to the RunMyProcess software automation platform.

INDEX WORDS: Text Classification, API Documentation,

Towards API Documentation Information Extraction: Challenges And Approaches

by

Mehdi Assefi

M. Sc., Montana State University - Bozeman, MT, 2016

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2021

Towards Information Extraction From Large Text Corpus: Challenges And Approaches

by

Mehdi Assefi

Approved:

Major Professor: Hamid R. Arabnia

Committee: Thib Taha
Khaled Rasheed
Sa'ar Hersonsky

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
December 2021

This work is dedicated to my parents.

Acknowledgments

There were many people who helped me along the way with my PhD journey, and most of all, I want to express my gratitude to Dr. Hamid Arabnia, without whom this dissertation might never have been finished.

I want to thank all of my committee members, each of whom gave me inspiration during this process. Thank you, Dr. Thiab R. Taha, and Dr. Khaled Rasheed, and Dr. Sa'r Hersonsky for generously advising me and for your kind support. I can not possibly thank you for your encouraging, thoughtful, and wise words. My gratitude to you is immense.

I am grateful to Dr. Juan Gutierrez for giving me insightful advice and for offering help and support when I needed it.

I want to thank my parents, my lovely brothers, Omid and Hadi, who have always supported me in all my endeavors, and my lovely sisters, Toktam and Hanieh, for their support.

Lastly, I want to thank Sanaz, my wife. I could not have done this without you.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Introduction

Application Programming Interface (API) consists of commands, functions, and protocols that are built into software programs so that they can automatically communicate with each other. Such a tool allows programmers to use predefined functions for system interaction instead of writing them from scratch for each application. Open API Specification (OAS)[1] further allows programmers to outline an entire API by describing all available endpoints, the operations that can be performed at each endpoint, and the parameters necessary to perform each operation (plus other related information, such as security and authentication). OAS therefore enables APIs to be totally self-descriptive. The objective of this study is to boost the productivity of software engineers by replacing unstructured API

[1] A sample OAS file and main schema is available at: https://swagger.io/docs/specification/basic-structure/that is also known as *Swagger Specification*

documentation with OAS, which is conducive to automated discovery and development. To achieve this goal, we automatically extract OAS from the wide variety of API documentation available over the internet and use this collected data to design natural language processing models which can then map descriptions of API functionality into machine-readable formats supporting greater automation and efficiency in the development of API-enabled software systems [Bahrami et al., 2018] [Bahrami and Chen, 2020].

1.1.1 Contribution

The main challenge of our study is to extract API features, such as endpoints, from heterogeneous API information and to map them to OAS-compliant formats, thereby providing a unified model for API specifications. Software engineers spend a significant amount of time understanding the documentation accompanying programs. System automation uses APIs to interact with each other. However, the API documentation of a provider can be anything from a single-page HTML description to a complex structure with information spanning several pages. Understanding this wide variety of API documentation structures and styles is therefore a labor-intensive and error-prone task for engineers. By providing a machine-learning platform that can extract and standardize API usage information, we believe we can accelerate the creation of API-enabled systems through automated learning. If we read the API documentation from even the most popular API providers, both the terminology and the structure of the documentation contain inconsistencies. Producing software programs to process such a wide variety of

API documentation styles is a challenging and demanding task in this era of automated coding. We need advanced machine-learning algorithms to accommodate for this variety and a unified specification for each API. We applied advanced machine-learning algorithms to address the variety of styles, formats, and terminology found in API documentation and to generate a unified API specification. Our study introduces several new developments in the mining of API documentation: i) developing a content-based learning model to predict the type of context for each block of API documentation; ii) developing a signature-based learning model to process the metadata for an API description by utilizing the heading HTML tags as a unique signature for recognizing an individual OAS object; iii) developing a Deep Mapping Model to specify a fine-grained OAS object type; iv) performing extensive experiments to validate trained models; v) implementing proposed components in real-world scenarios and measuring the performance of models on a variety of extracted API features; vi) collecting a sheer amount of API documentations, using our proposed classifiers to generate thousands of OAS-based APIs, validating OAS files by performing API calls, and finally deploying these validated APIs into our cloud-based, visual programming platform. To the best of our knowledge, our work is the first to apply signature-based learning and deep-content mapping to generate OASs. The rest of my dissertation is organized as follows. In chapter 2, we explore the relevant terminology, background, and related works. We perform a survey on the state of the art feature engineering and machine learning models that are suitable for our study. In chapter 3, we run an experimental study on Apache Spark MlLib - our chosen platform for building the

ML based Web crawler - by performing several classification and clustering models over different large datasets and under different hardware setups. We analyzed processing time and the accuracy of the models against Weka library (Version 3.7.12)[Bouckaert et al., 2010]. In chapter 4, we present our proposed AI-based web crawler that is designed to scrape the REST[2] API documentation over Internet. We provide a detailed description of the ML models that the crawler uses to identify the REST API documentation and collect them in a database. The collected data, which included 1,521 different REST API documents, is used as our training data for the models that we describe in chapter 5. Chapter 5 describes different components that we designed to achieve a unified API specification by training ML models based on a large set of API documentations which our API crawler collected. We explain the architecture of the proposed approach and our platform for extracting and mapping major API attributes to a specification. Our model design in chapter 5 includes three different approaches: training the context of HTML tables based on API attributes, using heading HTML tags for OAS objects, and applying a deep learning model to map each part of documentation to a fine-grained OAS type (API attribute). We also show the implementation of the proposed components for real-world APIs and the performance of each component as well as validation and deployment. We conclude our work in chapter 6.

[2]A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding[Fielding et al., 2017]

Chapter 2

Background and related work

Chapter Overview

Each NLP task may need several pre-processing, model design, and evaluation techniques. Pre-processing involves initial steps to prepare data for the models. Pre-processing text simply means to bring the text into a form that is predictable and analyzable for the ML task. A task here is a combination of approach and domain. A variety of machine learning and deep learning models can be used for text mining. In our study, we deal with several datasets which vary in properties, dimensionality, and size. Choosing the right method for pre-processing and feature engineering of the data and picking an efficient ML model is a crucial step that directly affects the accuracy and performance of the study.

In this chapter, we perform a survey study on several techniques of data pre-processing as well as machine learning and deep learning models for natural lan-

guage processing and present an overview of the related works in this area. Our goal is to analyze and compare different models and find the best fit for individual components of our project.

2.1 Feature extraction

In machine learning and statistics, feature selection which is also known as variable selection, attribute selection or variable subset selection, refers to the process of identifying and selecting a subset of relevant features or variables to be used in model construction. Feature selection helps to reduce the number of input variables to reduce the processing time while guarantees the maximum information gain.

2.1.1 General Feature Selection Techniques

Ranking-based Methods Ranking-based or filter methods consider each features' rank as the main criteria for feature selection. In this approach, we assign score to features based on their relevance. A threshold is used to retain useful features and remove the insignificant. Every feature contains information about different classes in the dataset to which it belongs. Features can be ordered based on their ability to discriminate among different classes. Filter methods are very popular and easy to implement because of their simplicity and the high performance of this group of algorithms as noted by users[Chandrashekar and Sahin, 2014, Sánchez-Maroño et al., 2007]. Feature relevance is the basic foundation of

this group of algorithms and has received a lot of attention in the research community over the past few years[Blum and Langley, 1997, John et al., 1994, Kohavi and John, 1997, Langley et al., 1994]. The definition of a feature's relevance is the key for different filter methods. One popular definition for the feature relevance relies on conditional representation of class labels by features. Based on this definition, a feature that is independent from some labels under certain conditions is considered irrelevant[Law et al., 2004]. Features' correlations are used to identify unique features. Based on Pearson's definition[Battiti, 1994, Guyon and Elisseeff, 2003], a correlation coefficient is defined as:

$$R(i) = \frac{cov(X_i, Y)}{\sqrt{Var(X_i) * Var(Y)}}$$

where x_i denotes the i_th variable from the input data with the dimension of $[x_ij, y_k]$, and Y represents the associated output, $cov()$ is the covariance, and $var()$ is the variance. Feature ranking using this method has the limitation of only identifying linear dependencies. Mutual Information is another method for feature ranking and based on entropy[Forman, 2003, Kwak and Choi, 2002, Lazar et al., 2012].

In practice, algorithms use one or a set of classifiers to identify an optimal subset of features. We should keep in mind that this subset is not unique, and similar results can be achieved by using different subsets of features. Yu *et al.* proposed a definition based on Markov blanket[Koller and Sahami, 1996] and grouped features as either irrelevant, weakly relevant and redundant, weakly relevant and non-redundant, or strongly relevant[Yu and Liu, 2004]. The authors of this paper

then categorized feature selection algorithms based on relevance and redundancy.

Wrapper Methods The general goal for feature selection is to obtain an efficient clustering with a minimum number of features. One intuitive solution is to use a brute force approach. Since with N features there will be 2^N possible solutions, and this yields an $NP-hard$ problem, one heuristic solution is to obtain clustering using a reduced feature set and then to calculate the object function by adding extra features to discard irrelevant features[Talavera, 2005].Pena *et al.* proposed a similar idea for feature selection in unsupervised learning of Gaussian networks[Talavera, 2005]. Genetic Algorithm and Particle Swarm Optimization are also invoked to reduce the time complexity of wrapper algorithms[Goldberg, 1989, Kennedy, 1995]. Wrapper algorithms generally follow two different methods to find the optimal feature subset Sequential selection algorithms build the feature subset by continuously adding features to an initially empty set or by removing features from the initially full set until the maximum objective function is achieved, while heuristic methods calculate the objective function with different combinations of features to achieve the maximum objective function.

Embedded Methods Embedded methods aim to minimize the computation time taken by evaluating different features' subsets in a wrapper approach. The most intuitive method to achieve this goal is by following a greedy approach[Battiti, 1994]. The objective function would be to maximize the *Mutual Information* between the feature and the class output while keeping the *Mutual Information* between the newly selected feature and the current feature set minimal[Chandrashekar and Sahin, 2014]:

$$G = I(Y, f) - \sum_{s \in S} I(f; S)$$

In this formula, Y represents the output, while the heretofore selected feature subset is denoted as S; S represents the feature set, and f represents the current feature that is being evaluated. The parameter β represents the importance of the mutual information between f and S. The function is applied to a Neural Network Classifier and results in a better feature subset[Chandrashekar and Sahin, 2014].

Evolutionary Computation Approaches A Genetic Algorithm can be implemented by considering each feature a binary bit in the chromosome. This algorithm can be designed to find the global maximum as the optimum subset of features that maximizes the predictor's performance[Alexandridis et al., 2005]. Cordón *et al.* used a modified version of GA called CHCGA[Eshelman, 1991] for feature selection[Cordón et al., 2006]. The CHCGA, which is a non-traditional version of Genetic Algorithms, has the following distinctions:

- CHCGA chooses the best N individuals from the set of parents and offspring, i.e. better offspring replace less fit parents.
- A half-uniform crossover function is being used which crosses over exactly half of the non-matching alleles.
- Parents are randomly selected without replacement during the reproduction phase. Parents with a Hamming distance less than the threshold will not mate. The threshold is dynamic, and in the case that no offspring generates, they will be decreased by one.

The CHCGA converges very fast and performs a more effective search by maintaining diversity[Chandrashekar and Sahin, 2014]. The number of calculations required to obtain an optimum feature subset is high due to the need for creating a new model for every subset evaluation. In the case of working with a large number of samples, training the predictor takes a huge amount of time. There is another drawback with using GAs, which is needed for evaluating the same feature subsets several times because the classifier results are not recorded for future use. Potential overfitting is another problem with using the classifier performance as the objective function[Kohavi and John, 1997]. Such happens when the classifier model performs poor generalization due to learning data so well as to overfit the learning. In order to avoid high accuracy with poor generalization power, a separate test set can be used[Kohavi and John, 1997].

2.1.2 Feature Selection Methods for Text Classification

An increasing amount of digital documents, such as scientific articles and social media posts have been daily generated in a large-scale fashion. Thus, categorizing and classifying of this wealth of data for different tasks has become very important.

In text processing, documents are traditionally represented as bag-of-words[Salton and Yang, 1973], where each individual term in a document is considered as a separate dimension or feature. Therefore, a document will be represented by a multi-dimensional feature vector where each dimension is associated with a weighted value of the term within the document. This weighted value is calculated using

tfidf which stands for *term frequency inverse document frequency*. Hence a corpus of even moderate-sized documents generates hundreds of thousands of dimensions. One of the most important issues in text clustering is therefore to deal with high dimensionality of the feature space[Bharti and Singh, 2015].
In this section we introduce popular feature selection methods in text classification.

Standard Methods

Odds Ratio Let $P(t|c)$ be the probability that t is the chosen word given assuming that the chosen word belongs to a class c. then we define

$$Odds(t|c) = \frac{P(t|c)}{[1 - P(t|c)]}$$

and the Odds Ratio is:

$$OR(t) = ln\frac{odds(t|c_+)}{odds(t|c_-)]}$$

Odds ratio favors features representing positive examples i.e. features barely occur in positive examples and never occur in negative examples will be assigned a comparatively high score. Hence there will be features which happen less frequent in the positive documents but relatively ranked at the top of the features list. This method works well with Naïve Bayes classifiers[Mladenić et al., 2004]

Information Gain Information Gain is a popular method in identification of

term importance. By definition, information gain of a term t is defined as follows:

$$IG(t) = -\sum_{i=1}^{|C|} P(c_i)log(P(c_i) + P(t)\sum_{i=1}^{|C|} P(c_i|t)log(P(c_i|t)$$

$$+P(\bar{t})\sum_{i=1}^{|C|} P(c_i|\bar{t})log(P(c_i|\bar{t}),$$

where c_i denotes the ith category, $P(c_i)$ is the probability of the ith category, $P(t)$ represent the probability of term t appear in the document and $P(\bar{t})$ represents the opposite. Information gain should be computed for every individual term in the corpus. Terms with IG less than a threshold will be removed from the feature space. This process requires the conditional probability of a category given a term as well as the entropy to be calculated. The probability computation has the time complexity of $O(N)$ and space complexity of $O(VN)$ with N representing the number of training documents and V is the size of vocabulary.The time complexity for entropy calculation is $O(Vm)$[Yang and Pedersen, 1997]

Chi-Square Chi-Square feature selection is a common method for feature selection with text data. χ^2 is a statistical test for testing the independence of two events. In the area of feature selection for text mining, this test indicates weather occurrence of an individual term and occurrence of a specific class are independent. With χ^2 in feature selection of text data, the score of each term in

document D is calculated as follows:

$$\chi^2(D,t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

where

- N represents the observed frequency and E represents the expected frequency,
- if term t is contained in the document e_t is assigned 1 and it is assigned 0 otherwise,
- if the document is in class c, e_c is assigned 1 and it is assigned 0 otherwise.

The χ^2 is calculated for each term and terms are rank based their score. χ^2 score of each term represents the null hypothesis H_0 which states that the document's class has no effect on the term frequency should be rejected and the occurrence of the class and term are dependent[Abbott, 2014]. The feature will be added to feature subset in this case. Evaluation Chi-square for feature selection on text data has been studied in the research community[Bahassine et al., 2018, Meesad et al., 2011, Zheng et al., 2004].

Combined Methods

Union of Feature Selections Union of feature selections(UFS) is basically a combination of standard feature selections without considering any standard. For example, a UFS algorithm can be created by the union of terms detected by Odds Ratio, Mutual Information, Information Gain, Chi-Square, and Document

Frequency. Therefore, vector dimension created by UFS is considerably higher than standard methods[Karaca and Bayir, 2017].

Correlation of Union of Feature Selections This method calculates correlation values between each term (v_t) obtained from UFS and classes (v_c) and stores the absolute values resulted by this calculation in a descending order. Then, CUFS is created as a result of selecting specific number of terms that have the highest values among them. For instance, correlation values of the terms resulting from UFS (100) were calculated; CUFS (100) was formed by choosing the first 100 terms with the highest values. Minimum vector dimension was reached through this method[Karaca and Bayir, 2017].

Sum of Term Frequency STF is like DF but utilizes a new method. DF deals with the number of documents where a term occurs while STF deals with the frequency of term occurrences across the documents. With STF, term occurrence in a document becomes more significant, ensuring that dominant terms in a given class are emphasized. For instance, the word "game" is a sports term and occurs more frequently in sports documents. If the term is evaluated once as it is in DF, effect of the term for sports class will be decreased. Process of term determination in STF is carried out with SQL query as in DF[Karaca and Bayir, 2017].

2.2 Deep Learning in Text Mining: Definitions, Benefits, and Challenges

Deep Learning is defined as a class of machine learning algorithms and techniques that utilize multiple layers of non-linear information processing to perform supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification. [LeCun et al., 2015, Ngiam et al., 2011, Schmidhuber, 2015].

Every deep learning approach is based on artificial neural networks(ANNs). ANNs increasingly receive new algorithms and growing amount of data that yields greater performance and better efficiency. The training process is called deep because it covers a large number of levels.

How it works? Every deep learning process consists of two main stages: training, and inferring [Ambrose et al., 2010, Schmidhuber, 2015]. During the training phase, large amounts of data is being labeled and their matching characteristics are determined. These characteristics are compared and memorized to make correct conclusions when facing similar pattern next time. During the training process, ANNs ask a set of true/false questions and classify data according to the answers received. The algorithm then labels the classified data. New labels can be assigned to the data during the inferring phase.

What makes Deep Learning and Machine Learning Different? Deep Learning is actually a descendant of traditional machine learning. The following list summarizes important differences between classical machine learning and deep

learning [Bengio et al., 2009, Goodfellow et al., 2016, Nielsen, 2015]:

- Deep learning requires a large amount of unlabeled training data to make a comprehensive decision while traditional machine learning can conclude using considerably smaller amount of data.
- Deep learning needs much higher-performance hardware.
- Feature identification should be performed by users before applying traditional machine learning algorithms, while deep learning creates new features on it's own.
- Traditional machine learning algorithms work by dividing a task into small units and then combine the results to make the final conclusion, while deep learning works on an end-to-end basis.
- The training process of deep learning algorithms takes longer than other machine learning models.
- Machine learning provides transparency in it's decision making which is not the case with deep learning.
- Deep learning is able to generate new features from features located in the training data-set without human supervision.

2.2.1 Text Mining

An increasing amount of document data, such as document literature, scientific articles, clinical notes, and online content – including drug reviews and heath

related social media posts – have been daily generated in a large-scale fashion. However, identifying hidden document knowledge from this wealth of data remains an important challenge. Deep learning methods have shown as a significant promise for discovering knowledge from a large corpus of text in an unsupervised manner [Habibi et al., 2017, Holzinger and Jurisica, 2014, Rios and Kavuluru, 2015]. Text mining has become an important part of many large-scale data analysis tasks like gene sequencing [Aerts et al., 2006], Human symptoms–disease network [Zhou et al., 2014], generation of curated databases [Li et al., 2015], and drug repurposing [Wang and Zhang, 2013]. The core task in document text mining is name entity recognition(NER) [Leaman and Gonzalez, 2008, Nadeau and Sekine, 2007].Entities can be proteins, species, diseases, chemicals, etc. Feature engineering i.e. determining a subset of features that best perceives entities of a certain class from other classes is an important step before performing text analysis method. Feature engineering is currently more of an art than a science and is a mostly a costly trial-and-error process. NER algorithms usually use whitelist and blacklist dictionaries which are also hard to construct and maintain [Leser and Hakenberg, 2005]. These solutions usually leads to very specialized tools that can not be generalized to be used with other entity types.

In text mining domain, there are three contemporary developments that lead to a significant improvement:

- **Word embedding** is a technique for representation of words by a low-dimensional vector that captures the frequencies of co-occurring adjacent words. Compared to bag-of-words – a traditional method of embedding –

this method is able to capture semantic similarities between words. These similarities are represented as mathematical closeness between vectors associated with words. For example, the words 'seize' and 'capture' are different from a syntactic point of view but they have similar meaning that leads both of them to a similar co-occurring words, while 'instance' has a completely different set of occurring words.

- **Artificial Neural Networks** are able to learn non-linear combinations of features and deep neural networks, and specially ***Long Short Term Memory(LSTM)*** performs this task in a very efficient way by memorizing long-term dependencies between words [Li et al., 2017, 2016, Limsopatham and Collier, 2016a,b].

- **Word2vec** which was presented by Google in 2013 is a semantic learning framework that employs ***Deep Neural Networks*** to learn distributed representations for words [Mikolov et al., 2013a,b,c]. Word2vec is very fast comparing to other modeling tools. many of Word2vec applications use cosine similarity to measure distance between words [Huang, 2008]. Word2vec is actually a group of shallow neural networks producing word embeddings. Word2vec is able to associate each word with a vector of a several hundred elements representing the relationship between word to other words. This vector is actually the hidden layer in neural network. We can use Word2vec in knowledge discovery applications by recognizing terms that are related to a specific term. Word vectors suggest that there are semantic or linguistic relationships between terms. Using these relationships, we can find:

- Closest term to meaning,
- Similarity of two terms,
- Dissimilar terms,
- Discovering terms by providing terms that are known to be related to the context. Example: *Man is to king is like woman is to___?(queen)*

Word embedding and LSTM are not new! The idea of representing words 'by the company they keep' is a very old concept in linguistics, usually called distributional semantics. This idea has gained a lot of attention recently and due to the novel idea that the embeddings are automatically tuned [Mackin, 1978]. LSTM was presented by *Hochreiter et al.* in 1997 [Hochreiter and Schmidhuber, 1997] and recent advances in computational capability and available data made it very popular and practical [Pascanu et al., 2013].

2.3 Document Text Mining: The state of the art

Extracting entities and their associated relations from document text data has attracted a lot of attention during past few years [Wei et al., 2016] and a lot of different applications have been proposed using this: protein-protein interaction(PPI) discovery [Pyysalo et al., 2007], drug-drug interaction discovery(DDI) [Segura-Bedmar et al., 2013], the bacteria biotope (BB) assignment [Delėger et al., 2016], and dverse drug event discovery(ADE) [Gurulingappa et al., 2012, Tafti et al., 2017c].

ADE aims to identify references of drug and diseases entities and find possible

ADE relations between them. Given a sentence "The patient was treated for **hemophilia**$_{disease}$ with **desmopressin**$_{drug}$ developed **sleeping disorder**$_{disease}$, the output of ADE discovery will be three entity mentions and ADE relation {**desmopressin**$_{drug}$, **sleeping disorder**$_{disease}$}.

Entity relation extraction is traditionally performed using two-step pipeline models. At the first step, entities are identified using name entity recognition(NER) technologies which usually work as a sequence labeling problem by using conditonal random fields(CRFs) [Finkel et al., 2005]. At the second step, each entity pair is inspected using classification models like support vector machine(SVM) [GuoDong et al., 2005] to find out if there is any task-specific relation between them. Pipeline models are very popular in the document community [Airola et al., 2008, Fundel et al., 2006, Kang et al., 2014, Nguyen and Tsuruoka, 2011].

Traditional methods generally suffer from two major problems [Li et al., 2017]:

- The errors generated in the first step - name entity recognition - can spread in the second step - relation classification. For example, if a disease or drug entity reference is incorrectly recognized, its related ADEs will be identified incorrectly.

- Second, there is not any interactions between two steps! These interactions may be helpful for the identification process. In some instances, entity recognition is not possible without getting help from the second step [Li and Ji, 2014].

Due to these limitations, new models process both steps at the concurrently.

Doing so, the model can reduce the error propagation problem and also parameters that are being shared by two sub-tasks help with the interaction between them. An attempt to build a framework to extract entities and relations based on linear programming has been made by Roth *et al*[Roth and Yih, 2007]. Li *et al.* proposed a single transition-based model to perform entity extraction and relation classification at the same time [Li and Ji, 2014].
The aforementioned solutions is that they are feature-based and require extensive feature engineering and they mostly suffer from feature sparsity. These approaches have a combined feature set from two joint tasks and this combined set is significantly larger than the ones with separate sub-tasks.

Deep learning has grabbed a lot of attention in the research community during past few years [Bengio et al., 2015, LeCun et al., 2015]. Deep neural networks acquire low-dimensional dense embeddings to indicate features which can resolve the feature sparsity problem. On the other hand, neural networks need less feature engineering since they learn features automatically and through training.

Ma *et al.* [Ma and Hovy, 2016] and Lample *et al.* combined recurrent neural networks(RNNs) and CRFs and resulted a framework with the best results on several datasets. RNNs and CNNs are two deep learning methods used for relation classification that are being widely. Authors used these models to learn relation representations between two entities or relations between two entities based on the shortest path dependency of them. An end-to-end relation discovery model is proposed by Miwa *et al.* which resulted promising performances in several datasets [Miwa and Bansal, 2016]. A transition-based feed-forward neural network

is used by Li *et al.* to extract drug-diseases entity reference and also their adverse drug events relations [Miwa and Bansal, 2016].

And finally, Fei Li *et al.* proposed a deep learning model for extracting document entities and their relations[Li et al., 2017]. Authors used CNN to encode character information of words into character level representation and then used a long short-term memory(LSTM) network to get the resulted character-level representation, word embeddings, and part of speech embeddings and learn the representation of the entities and their contexts. The proposed model improved precision and recall of the drug-disease entity recognition by 3.2 and 7.1%. The method also improves recall and precision of the ADE by 3.5 and 12.9%. This model needs less feature engineering due to the fact that it uses deep learning and it also benefits from information sharing between two steps.

Chapter 3

Large scale data processing - an experimental evaluation

Chapter Overview[1]

Artificial intelligence has been used in many ways by the research community to turn a variety of diverse and even heterogeneous data sources into high quality facts and knowledge, providing premier capabilities to accurate pattern discovery. However, applying machine learning strategies on big and complex datasets is computationally expensive, and it consumes a very large amount of logical and physical resources, such as data file space, CPU, and memory. A sophisticated platform for efficient big data analytics is becoming more important these days as the data amount generated in a daily basis exceeds over quintillion bytes.

[1]This chapter is based on the previously published article: Assefi, Mehdi, et al. "Big data machine learning using apache spark MLlib." 2017 ieee international conference on big data (big data). IEEE, 2017.

Apache Spark MLlib is one of the most prominent platforms for big data analysis which offers a set of excellent functionalities for different machine learning tasks ranging from regression, classification, and dimension reduction to clustering and rule extraction. In this contribution, we explore, from the computational perspective, the expanding body of the Apache Spark MLlib 2.0 as an open-source, distributed, scalable, and platform independent machine learning library. Specifically, we perform several real world machine learning experiments to examine the qualitative and quantitative attributes of the platform. Furthermore, we highlight current trends in big data machine learning research and provide insights for future works. Results of our experiments help to understand different capabilities of Apache Spark which we use as platform to build several components of our project.

3.1 Introduction

Digital datasets have been rapidly growing in size and complexity, and the large volume of daily generated data exceeds the boundary of normal processing capabilities, forcing us to take advanced computational infrastructures able to tackle parallel and distributed processing. Efficient mining of such massive data amounts is an extremely challenging practice which requires developing more sophisticated platforms to get extensive big data analysis accurately done in a timely fashion. Big data infrastructures have emerged to address the problem of big data analytics with the use of fast, reliable, and scalable computational architecture, providing

excellent quality attributes including elasticity, availability, and resource pooling with the ability of on-demand and ease-of-use self-services [Abbasi et al., 2016, Agrawal et al., 2011, Parashar et al., 2010, Talia, 2013]. Several big data machine learning frameworks are now available which, aside from lending practical contributions to other scientific disciplines, have demonstrated successful application in healthcare informatics [Archenaa and Anita, 2016, Fang et al., 2016, How et al., Lv et al., 2016, Pita et al., 2015, Tafti et al., 2017d, Van Horn, 2016], genomic data analysis [Ding et al., 2016, Masseroli et al., 2015, Syed, 2016, Wiewiórka et al., 2014], text mining [Garcıa-Pablos et al., 2015, Rong et al., 2011, Ryan, 2016, Tafti et al., 2017a], and stochastic modeling [Bhat et al., 2016, Ji et al., 2016, Sparks et al., 2013] purposes just to name a few.

Apache Spark MLlib is one of the most highly demanded platform independent and open-source libraries for big data machine learning which benefits from distributed architecture and automatic data parallelization. Apache Spark MLlib has been provided with Apache Spark [Zaharia et al., 2012], and it offers a set of dominant functionalists for a variety of machine learning tasks, including regression, dimension reduction, classification, clustering, and rule extraction. While machine learning and its effective applications have been studied for a long time in the research community, the study of big data machine learning libraries, such as Apache Spark MLlib has been very limited so far. This contribution is perhaps the first work that leverages a big data machine learning library, the Apache Spark MLlib 2.0, to tackle the problem of big data analytics. An objective of big data analytics is to get advanced computational infrastructures so that large-scale

data can be mined and analyzed in a timely and efficient manner. This constitutes the main motivation of the present work. Since big data analytics is computationally intensive, the performance and user experience are impacted by different hardware and/or software configurations. In this chapter, we evaluate the impact of different hardware and software configurations with a set of big data analysis tasks. Based on the findings of this study, we provide insights for future big data machine learning-oriented hardware, software, and model design.

The mechanism this chapter will discuss is to present, from the computational perspective, the Apache Spark MLlib 2.0 and its capabilities and advantages to big data machine learning. We demonstrate through extensive experiments the benefits of such a highly scalable machine learning library, and highlight the insights that can be drawn by big data analytics. Using several datasets including tens of millions of data records, we demonstrate that we can reliably utilize Apache Spark MLlib for a variety of large-scale machine learning strategies ranging from big data classification to big data clustering and rule extraction. We briefly summarize our main contributions in the following:

- We utilize Apache Spark MLlib 2.0 to initiate a study of big data machine learning on massive datasets including tens of millions of data records. The first and main contribution of the chapter is to introduce an exciting but challenging area to the machine learning community. While machine learning and its applications in research and industrial communities have been studied for several years now, the study of big data machine learning has been very limited so far. The present work is expected to bridge the gap

between the two areas, opening the doors for different interesting directions from the big data community to the fast-growing machine learning application area.

- We perform several large-scale real world experiments to examine a set of qualitative and quantitative attributes of Apache Spark MLlib 2.0. Furthermore, we establish a comparative study with Weka library Version 3.7.12 running on hadoop-2.7 [Bouckaert et al., 2010] - a very well-known Java-based machine learning library which has been widely used in the community. We evaluate multiple common big data machine learning models, including classification and clustering on real data, and compare their performance on various hardware and software configurations.

The rest of the chapter is organized as follows. We begin by introducing Apache Spark MLlib in Section 2. Section 3 explains the materials and methods which consist of a list of Apache Spark MLlib 2.0 components we investigate, plus several large-scale datasets. We then delve into the experimental validations in Section 4. Finally, Section 5 concludes with a summary of our findings and a discussion of several possible directions for future work.

3.2 Apache Spark MLlib 2.0

Apache Spark [Zaharia et al., 2010] as a highly scalable, fast, and in-memory big data processing engine, has been originally developed in the AMPLab at UC Berkeley, and it offers an ability to develop distributed applications using Java,

Python, Scala, and R programming languages. It comes with four major libraries, including Apache Spark Streaming, Apache Spark SQL, Apache Spark GraphX, and Apache Spark MLlib [Zaharia et al., 2016]. While Apache Spark Streaming as the core scheduling module of Spark, implements stream processing within highly fault tolerant and batch analytics architecture, The Apache Spark SQL implements relational queries to mine different database systems, introducing a data abstraction model called DataFrames [Frampton, 2015, Zaharia et al., 2012]. Apache Spark GraphX is a graph processing library on top of the Apache Spark which provides distributed computational models to process two common data structures, such as graph and collection. Apache Spark MLlib is a big data analytics library which provides more than 55 scalable machine learning algorithms that benefit from both data and process parallelization [Meng et al., 2016b]. The library includes implementation of a variety of machine learning strategies, such as classification, clustering, regression, dimension reduction, and rule extraction which enables easy and fast in-practice development of large-scale machine learning applications. Apache Spark MLlib also offers a set of multi-language APIs to evaluate machine learning methods, deploying several computational components that deal with optimization, latent Dirichlet allocation, linear algebra, and feature engineering pipelines [Meng et al., 2016b, Zaharia et al., 2012]. In the recent years, many facets of data science solutions have been amended by such a library [Armbrust et al., 2015, Liang et al., 2014, Lin et al., 2014, Shoro and Soomro, 2015, Sparks et al., 2015a,b], and many machine learning scientists and engineers have entered the development of new Apache Spark MLlib components to contribute

to the big data analytics community across the world. Figure 1 presents, from the development side, a pathway of the Apache Spark MLlib 2.0 in which the number of unique commits per release has been rapidly growing over the years. Apache Spark MLlib has been in very active development, and at the time of writing the related paper, the number of Apache Spark MLlib contributors was above 1000. Here, we briefly review the recent advances in Apache Spark MLlib applications. Tizghadam *et. al.* [Tizghadam and Leon-Garcia, 2015] proposed an open source, scalable platform called CVST. The system is proposed to be used in smart transport application development. CVST cosists of four major components for resource management, data dissemination, business intelligence, and application. The business intelligence component which is in charge of data analytics, uses MLlib to process the data and deliver it to front-end. Lee *et. al.* [Lee et al., 2017] proposed an architecture design of academic information system providing services for analyzing students' record patterns. The proposed recommender system uses Apache Spark MLlib to predict and recommend courses for the following semester. SparkText is a text mining framework developed by Ye *et. al.* [Ye et al., 2016]. The proposed system which employs Apache Spark machine learning and streaming methods along with Cassandra NoSQL database has been executed on a big dataset of medical articles to classify cancer types. Arora [Arora, 2017] analyzed mobile data collected from Web by using K-means algorithm on Apache Spark MLlib. The author offers an effective way of calculating the number of users of the network by clustering based on latitude and longitude values. Lee *et. al.* [Uddin et al., 2017] introduced ALMD - a feature descriptor by considering motion and

appearance and employing Apache Spark machine learning library random forest to recognize human activities. An attempt to generate a framework for analyzing the population structure using the next generation sequencing data is made by Hryhorzhevsk *et. al.* [Hryhorzhevska et al., 2017]. Authors of the paper proposed a distributed computing framework using ADAM combined with MLib, H2O and Apache SystemML. An architecture for automatic machine learning is proposed by Sparks *et. al.* [Sparks et al., 2015c]. The system is consisting of resource allocation estimator tunning, and optimizing components. The entire system is built upon Apache Spark and it leverages MLlib and other Spark components. The bigNN which is developed by Tafti *et. al.* [Tafti et al., 2017b] is another interesting big data analytics component implemented on top of the Apache Spark, and it is capable to tackle very large-scale biomedical sentence classification. There still exists a list of valuable contributions on big data analytics. Interested readers are referred to [Allahyari et al., 2017, Fazli et al., 2017, Gandomi and Haider, 2015, Gopalani and Arora, 2015, Lee et al., 2017, Lv et al., 2017, Pecaric et al., 2017, Raghupathi and Raghupathi, 2014, Shyam et al., 2015, Yousefi et al., 2015] for further readings.

3.3 Materials and Methods

In this section we further explain the materials and methods of the current research study. We shall begin with the Apache Spark MLlib components, and then introduce the datasets.

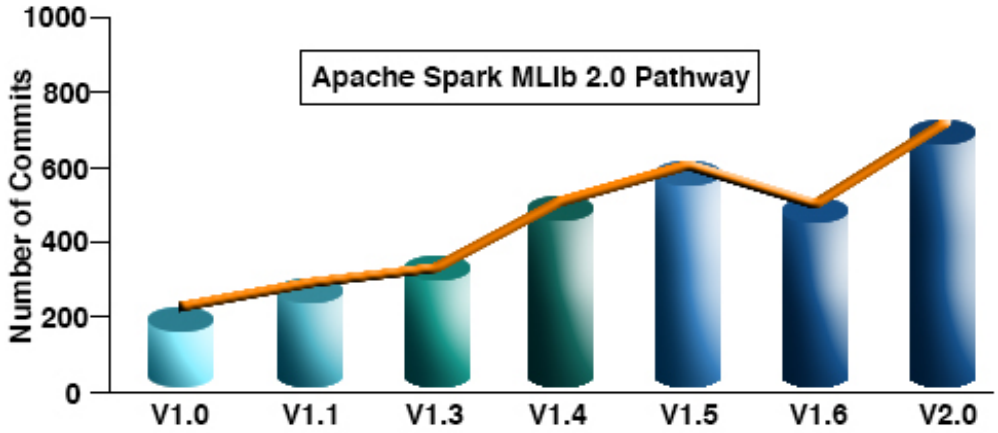


Figure 3.1: The development pathway of Apache Spark MLlib 2.0. DataFrame-based APIs in Java and Scala programming languages have been provided by MLlib V 1.2. The MLlib v 1.3 and v 1.5 came with Python and R APIs respectively. Unification of APIs (Dataset &DataFrame), built-in CSV file support, and rapid explorative data analysis offered by Apache Spark MLlib 2.0. It supports for Generalized Linear Models (GLM), Naïve Bayes, Survival Regression, and K-Means in R.

3.3.1 Machine Learning Components

To evaluate the ability of the Apache Spark MLlib 2.0 library in analyzing big data sets, we focused on a set of supervised (classification) methods, including SVM (Support Vector Machine), Decision Tree, Naïve Bayes, and Random Forest, along with a widely-used unsupervised (clustering) machine learning algorithm, namely KMeans. To address a comparative study, the machine learning algorithms in Apache Spark MLlib 2.0 library were compared to the same algorithm in Weka library (Version 3.7.12) running on hadoop-2.7. To work with Apache Spark MLlib 2.0 and also Weka components, We utilized Java2SE 8.1 programing language for all parts of the code and implementations. For each set of the ML algorithms, we employed the same configurations (e.g., regularization, cost, loss, kernel type, seed, etc.) with equal value of parameters offered by both Apache Spark MLlib 2.0[Meng et al., 2016a] and Weka libraries[Bouckaert et al., 2010].

3.3.2 Datasets

Six various big datasets were used to analyze and compare the Apache Spark MLlib 2.0 performance. Five datasets from the UCI Machine Learning Repository, and a dataset from the US government's Bureau of Transportation Research and Innovative Technology Administration (RITA) Web sites[Nambisan et al., 2014].

The first dataset called "HEPMASS" includes high-energy physics experiments to search for the signatures of exotic particles, and it is associated for binary classification task. The second one named "SUSY" is associated to a binary classification problem to distinguish between a signal process that produces su-

Table 3.1: Datasets' attributes

Dataset	**Characteristics**	**Attributes**	**Data records**	**Size**
HEPMASS	Multivariate	Real	7,000,000	2.20 GB
SUSY	Multivariate	Real	4,999,998	1.81 GB
HIGGS	Multivariate	Real	10,999,998	5.74 GB
FLIGHT	Multivariate	Integer, Nominal	47,113,991	3.09 GB
HETROACT I	Multivariate, Time-Series	Real	13,062,477	879 MB
HETROACT II	Multivariate, Time-Series	Real	13,932,634	977 MB

persymmetric particles and a background process that does not.The third dataset called "HIGGS" is related to a binary classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not.The forth dataset called "FLIGHT" which includes flight information from October 1987 to April 2008. There are variables related to flight time, flight origin and destination, flight elapsed time, arrival airport, delay time, and etc. within the dataset. We utilized this dataset for classification purpose. The fifth and sixth datasets called "HETROACT I" and "HETROACT II". These heterogeneity datasets for human activity recognition from Smartphone and/or Smartwatch sensors are designed to investigate sensor heterogeneities' impacts on human activity recognition algorithms, and we utilized them for clustering purpose. For classification purposes, we employed 75% of every dataset to train the classifier, and 25% to test, all with using 4-fold cross validation to get the Area under ROC (auROC). Table I illustrates the detailed attributes of each dataset.

Table 3.2: The employed configurations of two VMs on a VMWARE Cluster environment.

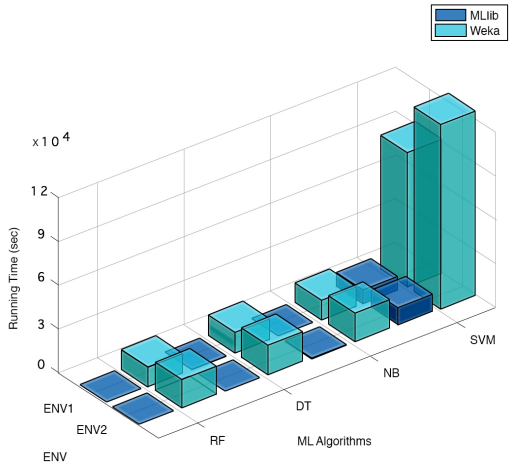
Environment	Number of nodes	HDD	RAM	CPU
ENV1	2	1 TB (in total)	8 GB (each)	4 vCPUs (each)
ENV2	2	1 TB (in total)	16 GB (each)	8 vCPUs (each)

3.3.3 Testbed Environments

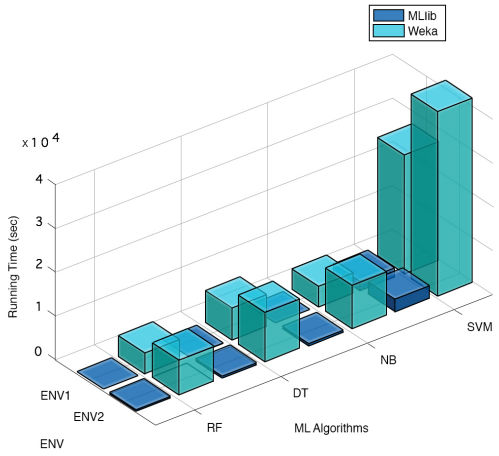
Two VMs in a VMWARE Cluster environment have been used to obtain experimental results. We utilized two different configurations namely "ENV1" and "ENV2" on both VMs as further illustrated in Table II. The 64-bit CentOS 6.8 operating system with Xeon E5-2690V3 2.6 GHz CPU were used on both VMs.

3.4 Experimental Results

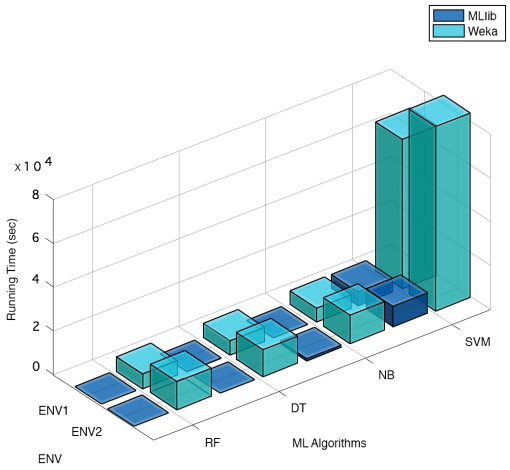
We focused on the following supervised (classification) methods: SVM (Support Vector Machine), Decision Tree, Naïve Bayes, and Random Forest. We also employed K-Means as an unsupervised (clustering) algorithm to perform experimental studies on both supervised and unsupervised machine learning methods. Here is a summary of our experiments over the six datasets illustrated in Table I. Results present the running time of Mllib and Weka with same hardware setup. The area under ROC(s) obtained by Weka and Apache Spark MLlib with the use of SVM, Decision Tree, Naïve Bayes, and Random Forest methods over four different datasets are presented in Table III. Table IV discusses the experimental results obtained by the K-Mean algorithm across the datasets. Figures 3.2a, 3.2b, 3.2c, and



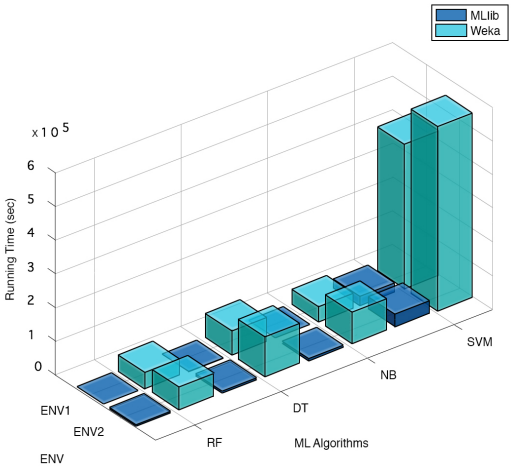
(a) The results obtained on the *Flight* dataset.



(b) The results obtained on the *HEPPMASS* dataset.

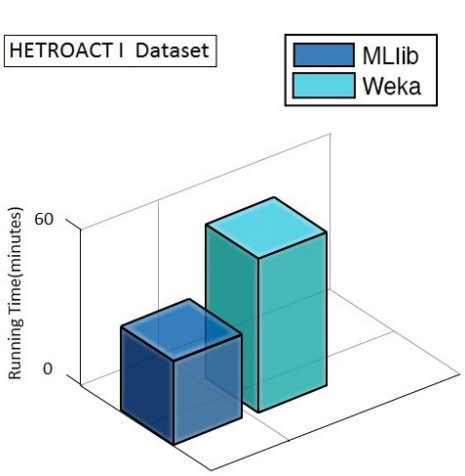


(c) The results obtained on the *HIGGS* dataset.

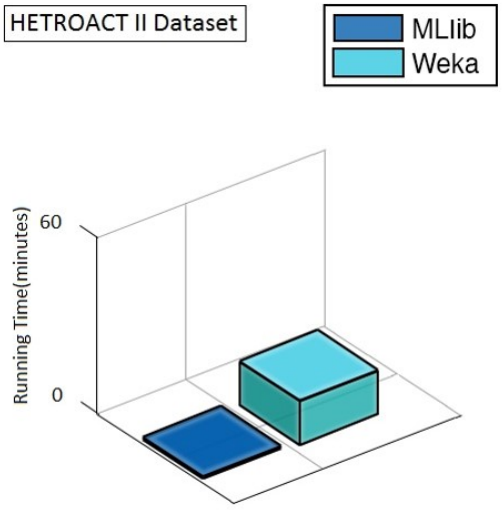


(d) The results obtained on the *SUSY* dataset.

Figure 3.2: Running time of Apache Spark MLlib compared to Weka under different classification experiments. NB stands for Naïve Bayes, DT stands for Decision Tree, and RF refers to Random Forest.



(a) Running time of K-Means clustering over the *HETROACT I* dataset.



(b) Running time of K-Means clustering over the *HETROACT II* dataset.

Figure 3.3: The running time of Apache Spark MLlib K-Means compared to Weka K-Means clustering component.

3.2d show the running time of 4 selected classification algorithms on the proposed datasets. The running times of K-means on HETROACT I and HETROACT II datasets is summarized in figures 3.3a and 3.3b. Summary of the experimental results are:

- Area under the ROC obtained by both Weka and Apache Spark MLlib are pretty similar to each other, and the difference is not statistically significant. The small differences between the Area under the ROC achieved by Weka and Apache Spark MLlib might be associated to the detailed parameters of each algorithm in randomly selecting train and test instances while we are doing 4-fold cross validation, or they could come from the very detailed

internal parameters of each algorithm.

- The comparative study demonstrates that the Apache Spark MLlib, as expected, is able to be faster in comparison with the Weka components we have utilized, and performing a t-test on the running time matched by either classification algorithms or the clustering method shows statistically significant differences (at $p < 0.01$) between Apache Spark MLlib and Weka.

hryhorzhevska2017scalable venkataraman2016ernest ma2015design

3.5 Discussion and Outlook

The amount of digital data being collected is expanding on a daily basis, and as a result, the science of data analytics and processing should be more technologically advanced to enable data scientists to turn this large body of structured and even unstructured data into high quality information and facts. Computer engineers and scientists have already entered the development of big data machine learning components to solve the problem of pattern discovery form large-scale data sources more efficiently, and Apache Spark MLlib is one of the widely-used big data machine learning library available to the community.

Apache Spark MLlib is a very strong tool for big data analytics and as results of our study shows, it presents spectacular performance in terms of running time. Weka, on the other hand, works slower than Apache Spark MLlib under big loads of data samples. Considering different file systems and configurations used by Apache Spark MLlib and Weka, this comparison may not be deemed fair though.

Table 3.3: Area under ROC associated with the classification algorithms performed by Apache Spark MLlib and Weka library.

Environment	**Dataset**	**Algorithm**	**MLlib**	**Weka**
	SUSY	SVM	0.7932	0.7829
		Random Forest	0.7614	0.7731
		Decision Tree	0.7528	0.7690
		Naïve Bayes	0.7853	0.8031
	HIGGS	SVM	0.5690	0.5543
		Random Forest	0.5680	0.5712
		Decision Tree	0.5829	0.5873
		Naïve Bayes	0.5741	0.5722
ENV1	FLIGHT	SVM	0.5189	0.5312
		Random Forest	0.5305	0.5581
		Decision Tree	0.5736	0.5819
		Naïve Bayes	0.5570	0.5384
	HEPMASS	SVM	0.9591	0.9543
		Random Forest	0.8948	0.8961
		Decision Tree	0.8995	0.8971
		Naïve Bayes	0.9114	0.9398
	SUSY	SVM	0.7932	0.7829
		Random Forest	0.7614	0.7731
		Decision Tree	0.7528	0.7690
		Naïve Bayes	0.7853	0.8031
	HIGGS	SVM	0.5690	0.5543
		Random Forest	0.5680	0.5712
		Decision Tree	0.5829	0.5873
		Naïve Bayes	0.5741	0.5722
ENV2	FLIGHT	SVM	0.5189	0.5312
		Random Forest	0.5305	0.5581
		Decision Tree	0.5736	0.5819
		Naïve Bayes	0.5570	0.5384
	HEPMASS	SVM	0.9591	0.9543
		Random Forest	0.8948	0.8961
		Decision Tree	0.8995	0.8971
		Naïve Bayes	0.9114	0.9398

Table 3.4: The general performance of Weka and Apache Spark MLlib K-Means algorithm.

Dataset	**MLlib**	**Weka**
HETROACT I	Time: 0 min, 48 sec	Time: 11 min, 40 sec
	Number of Clusters: 5	Number of Clusters: 5
	SSE: 1.2403975185657699E34	SSE: 1.2419981124641322E34
HETROACT II	Time: 19 min, 57 sec	Time: 53 min, 16 sec
	Number of Clusters: 5	Number of Clusters: 5
	SSE: 1.2312227116685513E35	SSE: 1.2619357121584791E35

We ran MLlib on Spark distributed file system and used Hadoop distributed file system for Weka. But our goal is to show how Spark performs with large sets and Weka is used here as a reliable baseline that is agreed by the research community. There are many features with Weka that Spark can not compete with: (1) There is a big pool of documents and resources available for users, (2) It is very straightforward and easy to use for non-expert users, and (3) It has a perfect graphical user interface. Weka supports a vast variety of machine learning algorithms.

Apache Spark MLlib offers fast, flexible, and scalable implementations of a variety of machine learning components, ranging from ensemble learning and principal component analysis (PCA) to optimization and clustering analysis. Apache Spark MLlib also offer options for distributed processing by parallel processing and support of big data tools that utilize distributed architectures. These criteria will decrease the processing time required and, at the same time, increase the time available to interpret analytics results. This becomes very important when the machine learning task has many predictions to calculate. The distributed architecture can also take advantages of some of the big data tool sets available

to help break apart the machine learning component to improve overall running time. Integration is another advantage of Apache Spark MLlib, meaning that the MLlib gains from several software components available in the Spark ecosystem, such as Spark GraphX, Spark SQL, and Spark Streaming, and a wide range of well-organized documentations, including code samples are publicly and freely available to the machine learning community.

Our future work will be focused on adding more practical experiments with most of the Apache Spark MLlib components by utilizing a variety of bigger datasets. As part of our future work, we are working to run an experimental evaluation of Apache Spark MLlib under a diversity of programming languages (e.g., Python and R), clusters and also hardware or/and software configurations, employing a set of big datasets with variety of characteristics within the data.

Chapter 4

An Intelligent Method for Construction of API Corpus at Scale

Chapter Overview[1]

The number and variety of Web APIs is growing rapidly. Each API provider offers API documentations which could be complex description or a single HTML page description. In order to understand APIs behavior and endpoints, software engineers read API documentations. Understanding the variety API documentations is a labor intensive and also error-prone process. In this chapter, we introduce an approach for the construction of a large corpus of API documentations. The

[1]This chapter is based on the article: Assefi, et al. "An Intelligent Method for Construction of API Corpus at Scale." IEEE Big Data 2021: CAS#6 - under review

API Corpus can be used for information extraction to understand a variety of APIs. We use a Spark-based web-crawler where different web-crawler agents work together to collect raw HTML pages. We employ machine learning to classify the content of API documentations. The proposed method classifies the content of API documentation and predicts each content as API reference page, RESTful API documentation page and irrelevant or non-RESTful API documentation. We also sub-classify the content of APIs per provider.

4.1 Introduction

API providers explain the API usage, API security authentication and other specific features through API documentations. Software engineers must read API documentations in order to interact with API and/or integrate an API into another systems. Some API providers offer a machine-readable format of API documentation. One popular API documentation format is named Open API Specification (OAS)[2]. This format is also known as *Swagger Specification* and is a structured-based API description of a REST (REpresentational State Transfer) API [Fielding, 2000]. Swagger format allows programmers to outline an entire API by describing available endpoints and operation on each endpoint, parameters needed for each operation, authentications, etc. OpenAPI allows APIs to be self-descriptive. However not all API providers offer OAS format of API documentations. In this study, we design a Spark-based [Assefi et al., 2017] carwler

[2]A sample OAS file and main schema is available at: https://swagger.io/docs/specification/basic-structure/

to collect API documentation corpus at scale that consists of a sheer number of API documentations. We introduce a machine-learning model that classifies the collected pages. The API Corpus can be used for a variety of purposes such as:

- Constructing Open API specification where a machine-learning model can automatically generate OAS file from API docs [Bahrami et al., 2018]
- Low-code environment where the API corpus can be used for training a model to produce source code to interact with APIs automatically,
- Learning API usage from API documentation.

This chapter organized as follows. Section 4.2 explains how we collect a list of APIs and the metadata of each API, then utilize a distributed method for collecting a large number of API documentations. We'll introduce deep learing models for page content filter and REST filter, and describe the techniques that we proposed for page annotation. However, the collected HTML pages may not only corresponds to API documentations and we may collect non-relevant documentations. Once API documentations have been collected, an API provider may offer different APIs. We describe API Scoping approach where it categorized API documentations into different APIs per API provider. In section 4.3, we describe the ML models for document classification. Section 4.4 covers the algorithms that we developed for API scoping. Section 4.5 covers the experiments and results and we summarize the chapter in section 4.6.

4.2 API crawling

The World Wide Web has grown from a few thousand websites in the early 1990's to about 2 billion websites today. Search engines rely on gigantic collections of web pages that are collected using web crawlers. Crawling is in fact the process of collecting data from web pages by following hyperlinks extracted from a small set of web pages for further processing[Castillo, 2005]. Web search technology has been studied and improved by researchers over the past few years and a significant amount of work has been done on the analysis of Web anatomy, crawling strategies, indexing, and ranking techniques[Arasu et al., 2001, Laura and Me, 2017, Sasaki et al., 2015, Sharma and Gupta, 2015].

As a major unit of API learning, the Web-crawler is in charge of collecting a large number of web pages that contain API documentations. We use a Web-crawler that is composed by 32 parallel agents each collecting a subset of API documentations from which the APICorpus is being built. Several sources are used by our Web-crawler to collect a comprehensive list of APIs: ProgrammableWeb[3], API Guru, API Harmony, RapidAPI, API Mashup, etc. The list of APIs includes at least one API title and one API documentation URL. Some sources may also include other metadata related to the API, such as the API website link or the API publisher link. Our pointer list consists of more than 20,000 APIs. Our target is REST APIs. In order to detect Rest APIs by the crawler, we add a machine learning component to it. Another machine learning-based model identifies if the page content is related to API documentation.

[3]https://www.programmableweb.com/category/all/apis

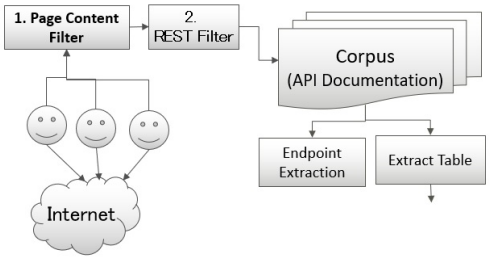


Figure 4.1: Models to filter out unrelated web pages and find REST API content for the crawler

We designed two models to improve performance of the crawler:

- In order to detect REST APIs, we add a machine learning component to the web crawler.
- A machine learning model is also designed to identify if the page content is likely to be API documentation, so the crawler collects just relevant pages using this model.

Figure 4.1 explains this process.

4.2.1 Page Content Filter

Our page content filter is in fact a Convolutional Neural Networks (CNN) model that is designed for sentence classification. In order to train the model, we used a training set containing positive sentences - the sentences that are related to an API documentation - and negative sentences - the sentences that are not related to any API documentation. The model's task is to perform classification on the

sentences and to identify the probability that a sentence is related to an API documentation. Each sentence in the dataset contain information about:

- How to call an API?
- What are parameters of an API?
- How much is the cost of calling an API?
- API policies,
- etc.

There are also sentences that provide similar information but they are not related to any API. For example, we may have extracted the sentences related to applications that do not offer any API. Figure 4.2 explains the process.The input is set of pre-processed API documentations. The input API docs may be given directly to the sentence extraction, or they need to be processed by the information extraction part to be organized in JSON format - the result can be sent to the sentence extraction component. The generated JSON files can also be manually annotated. Each file that is related to an API documentation will be labeled as ***"positive"***. Having the positive cases labeled, the machine is able to generate and label a reverse list containing the files that are not explaining API documentations, AKA ***"negative"*** cases.

The fully annotated cases are used to build and train a Convolutional Neural Network (CNN) which serves as our classification model. The classifier identifies the positive and negative sentences. Using this model, our web crawler is capable of

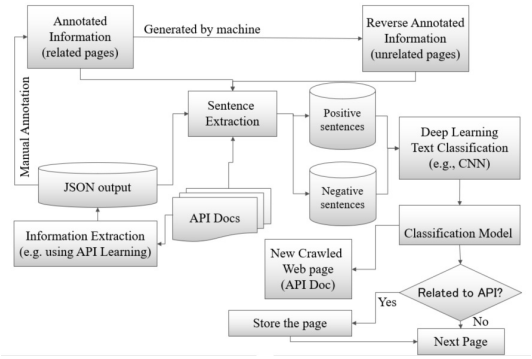


Figure 4.2: Page Content Filter

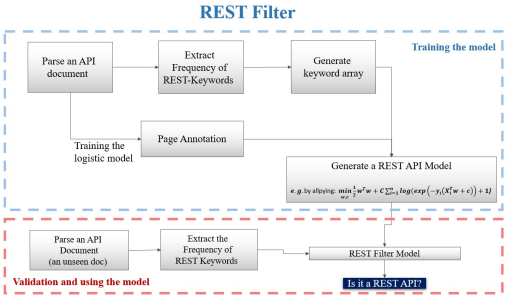


Figure 4.3: REST Filter

recognizing the web pages that are related to APIs. If a web page is API-related the crawler scrapes the page and stores the data in a local disk, the crawler will ignore the pages that are identified as unrelated to APIs.

4.2.2 REST Filter

Once a web-crawler collects data from the web it needs to analyze the content and identify if the content belongs to a REST API documentation. REST filter is the component that we designed to perform this task. In order to identify a

REST API document, the following steps must be followed by the REST filter:

Parse the API document

Once a web-crawler agent collects information from the web, it parse the page.

- If the page is using a JavaScript to load the data, REST filter uses a browser-based data collection component to compile the page and get HTML source file. Otherwise, it collects HTML page,
- Extracting text (i.e., English language sentences- but not limited to one language) from HTML page,
- Cleaning the text by removing stop words(e.g., the), signs (e.g., !) and etc.

Extracting Frequency of REST-Keywords

REST API has several keywords that has been used widely in API Documentations for this group of APIs. The set of REST API keywords include GET, POST, PUSH, DELETE, etc. The keywords are regularly written as capital words. This component counts the frequency of the REST keywords and identifies how many times each REST keyword appears in one document.

Generating the keyword array

Computing the frequency of each keyword and put all keywords and their frequency, generates a matrix of keyword and their frequency in one document.

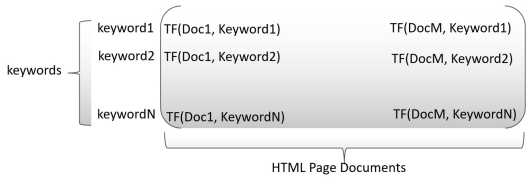


Figure 4.4: Term Frequency Matrix

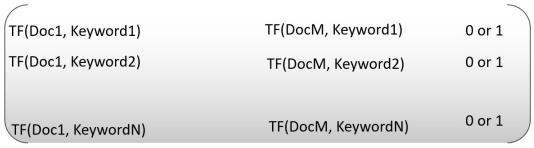


Figure 4.5: Page Annotation

Term frequency of keywords (the number of repetition of the given keyword in one HTML page document) is returned as TF value.

Figure 4.4 shows the structure of the resulted matrix.

Page annotation

- In order to train the REST filter model, we have to generate a set of annotations to show if an individual page is related to REST API.
- If the page is related, it is marked as "1"; Otherwise, it marks as "0".
- For each "doc" in the figure4.4, we will add this annotation. figure4.5 shows the resulted structure.

Generating a REST API Model

- In order to detect an API documentation or multiple documentations is REST API, we use logistic regression model. For example, a logistic regression can be defined as follows:
  $min_{w,c} \frac{1}{2} W^T W + C \sum_{i=1}^{n} log(-y_i(X_i^T W + c)) + 1$
- The model of a logistic regression as known as maximum-entropy classification is a linear model for classification.
- It takes X array as input, and Y array for training purpose to find maximum-entropy of X between [0-1].
- We consider a list of keywords of REST APIs, such as GET, POST, REST, PUSH, DELETE, etc.

Parsing an API Document (unseen doc):

- This is similar process to the first step but the web-page source is different because the page in this section is not appear in training dataset.
- The web-page (source) in this section is used for validation of the model to evaluate the results or it can be used to predicate if the page is related to REST API or not related.

Extracting Frequency of REST-Keywords

This component is similar to the second step. The only difference is its input that comes from the unseen document processing.

REST model

- The generated model provides a predication model for any input including seen input as appear in the training dataset or unseen input from training dataset.
- It returns True of False as its results if the page is related to REST API.

4.3 API Documentation Classification

Every text categorizing task aims to map documents into a predefined number of categories.The document may be assigned zero, one, or multiple categories by the classifier. The goal of using machine learning to perform the classification is to train the classifier using the existing samples and automatically perform the category assignments. This is a supervised learning task. The classification model treats each category as a separate binary classification problem. First step in every text classification task would be the document transformation. Documents are typically strings of characters and they should be transformed to a suitable representation to be used by the learning algorithm. Theory suggests that using word stems as representation units doesn't significantly affect the results of the classification and the ordering of these representation units in the document is negligible for many tasks[Joachims, 1998]! Attribute-value representation of text is usually used for the text processing. With the simplest representation each word w_i corresponds to a feature and the frequency of w_i in the document is considered as it's value. Considering all words in the document creates a large

feature vector. In order to avoid this problem just important words considered as feature. The representation method may decide to include the words with a minimum frequency. Stop words (and, or, etc.) may also be removed. This method of text representation creates a very high-dimensional feature space consisting of thousands of features. Using feature selection algorithms may result improved generalization accuracy, better performance, and also avoid overfitting. It has been shown that scaling the dimension of the feature vector using inverse document frequency improves the performance [Salton and Buckley, 1988].

Support vector Machines

Support vector machines are based on the Structural Risk Minimization principle[Vapnik, 2013]. Structural risk minimization aims to find a hypothesis h that reduces the probability of making error on an unseen and randomly selected test sample. In order to relate the true error of a hypothesis h with it's error on the training set and complexity of h - which is measured by VC-Dimension - an upper bound may be used. Support vector machines find the h that minimizes this bound by effectively controlling the VC-dimension of h.

One important property of SVMs is Independence of their performance from the dimentionality of the feature space. The compleaxitiy of SVMs hypothesis is measured based on the margin that is used for separating the line, not the number of features, which means that generalization is possible whenever the data is separable with a wide margin using the functions from the hypothesis space - even if there exist to many features.

Why SVM for Text Categorization?

In order to find the best method to build a classifier we need to review some important properties of text:

- **High dimensionality of feature space:** Text classifiers have to deal with large number of features(more than 10000). As we already know, SVMs apply overfitting protection which is not dependant on the dimentionality of feature space. This property gives SVMs the potential to handle the problem of large feature spaces.

- **Few irrelevant features:** In case of having a high dimensional feature space, one solution is to assume that many of those features are irrelevant and then use feature selection to reduce the features that are irrelevant. Unfortunately, this is not an effective solution in the text classification domain - there are very few irrelevant features! Experiments on text data show this fact by ranking features based on the information gain they provide and then comparing the information gain from low-ranked features against the higher rank ones. Results show that even low ranked features provide a considerable amount of relevant information[Joachims, 1998]. Even using those worst features to train the classifier results much better performance than random. In conclusion it is safe to say that a good text classifier has to combine many features(learn a *"dense"* concept) and any aggressive feature selection method may result in the information loss.

- **Sparse document vectors:** The number of unique tokens in a document

is a very small fraction of the total number of unique words of the corpus. This makes the document vectors sparse containing many more zeros than non-zeros! Theoretical and empirical evidence exist and show that SVM-like methods perform well for problems with sparse instances and dense concepts [Kivinen et al., 1997].

- **Linearly separable categories:** Experiments show that many categories are linearly separable. SVMs, on the other hand look to find linear(polynomial) separators.

4.4 API Scoping

Each API provider may offer several APIs and each of them have their own API specification. API web-crawler is required to cluster collected API documentations according to each API and their scope. API Scoping introduces a new approach to cluster multiple APIs of one provider into different clusters.
Let $\mathcal{A}_{i,j}$ be ith API of jth API provider. Therefore, $\mathcal{S}$ represents the scoping of all API providers and their APIs:

$$\mathcal{S} = \bigcup_{j} \bigcup_{i} \mathcal{C}_{i,j} \tag{4.1}$$

We take URLs of all collected API documentations and count each segment (URI). For instance, the following figure shows the total number of common URIs for each URL. In this figure, we use abbreviation to explain the example but a sim-

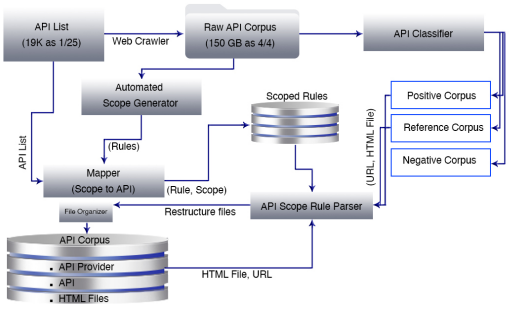


Figure 4.6: API Corpus Integration

ilar structure can be found from Twitter developer page[4]. One essential step in constructing the API scopes is to categorize the URLs based on their common prefix and sort them based on their majority. Consider "a.com/bc.html", "a.com/public/de.html", and "a.com/fg.html" as a simple example. In this example a common prefix "a.com" exists with 2 instances and another prefix - "a.com/public" - exists with i instance. This number is used to sort the extracted URLs and finally each API from list will be assigned a scope.

Figure 4.7 shows the process of API construction integration.Algorithms 2, 3, 4, and 1 show the details of this process.

Automated API Scoping We use similarity score between scope and API titles as measurement to map each scope to an API. API providers may offer several APIs. For the mapping purpose, we create two groups for each provider. First group includes different scopes and the second group contains all API titles from the same provider. The goal is to map items in the first group to the second.

[4]https://developer.twitter.com/en.html

Algorithm 1 Init XML Algorithm

```
X ← Positive class HTML pages from classifier
X_g ← GroupXByProvider(X)
countdict ← dict()
for each group_g in X_g do
    for each file in g do
        regex ← re.compile("/")
        prefixlist ← GetAllPrefixSplits(regex)
        for each prefix in prefixlist do
            countdict[prefix] ← countdict[prefix] + 1
        end for
    end for
    countdict ← SortedByValue(countdict, reverse=True)
    WriteToXML(g.provider, countdict)
```

Algorithm 2 Scoping Algorithm

```
α ← none : min edit threshold
β ← none : ngram value
for each group in g in XML do
    for for each rule in g do
        rule ← get_rule()
        apis_g, apiportal_g, homepageurl_g ← GetApiURLs(g.provider)
        α ← none
        scope ← UpdateXMLSubstringMatch(apiportal_g, rule)
        if (scope):
          WriteScopeToXML(scope, rule)
        else
          scope ← UpdateXMLSubStringMatch(homepageurl_g, rule)
    end for
end for
PropagateAllScopes(xml)
AdvancedScope()
=0
```

Algorithm 3 Scoping Algorithm continued: advanced scope()

```
1: function AdvancedScope():
2:     for each group g in XML do
3:         for each rule in g do
4:             rule,title,tagtext←GetRuleTitleTag(node.text,soup.title,'h1')
5:             tokens ← TokenizeClean(rule, title, tagtext)
6:             n_grams ← ngrams(tokens, β)
7:             apis_g ← GetApiURLs(g.provider)
8:             feasible_set ← None
9:             for each api in apis_g do
10:                tokens_api ← TokenizeClean(api)
11:                if n_gramMatch(n_grams in tokens_api):
12:                    feasible_set.append(api)
13:            end for
14:            if len(feasible_set) ==1:
15:                return feasible_set[0]
16:            else scope ←
17:        argmin(GetMinEditDistance(α, feasible_set.join(tokens))
18:            if scope
19:                WriteScopeToXML(scope, rule)
20:        end for
21:        PropagateAllScopes(xml)
22:        decrease β
23:        decrease α
24:        repeat
```

Algorithm 4 Processing HTML URLs

1: $X \leftarrow htmlpages$
2: Group X by provider to get K groups ($k = \#ofproviders$)
3: $Count\_Dictionary = \{\}$
4: **for** $filename(.html)$ *in each provider group* **do**

Regex $\leftarrow$ *re.compile(/) #search pattern, used to split url*

5: **for** each occurrence of "/" **do**
6: compute prefix
7: **end for**
8: Count_dictionary[prefix] += 1
9: **end for**
10: Reverse Sort by value (higher value corresponds to more basic url in a path structure)
11: Write each group to XML
12: Output rules = 0

For example, similarity measurement might suggest "fujitsu.com/ScanAPI" scope to be mapped to "Fujitsu Scanner API". Figure

Figure 4.7 shows the steps taken for the automated API scoping. API title is extracted from the API metadata and useless characters being removed. There are some other useless parts in API titles that are better to be removed. For example in the title "Fujitsu Scanner API", API has to be removed as it doesn't provide any usefull information. As every regular NLP task, the title has to be tokenized as well. The same process is applied on the entries of the positive REST API pages list which refers to pages that explains REST API documentations. We apply bigram and 3-gram for the sequence matching. In case of a match between an API title, the algorithm maps the scope to the URL. In order to improve the

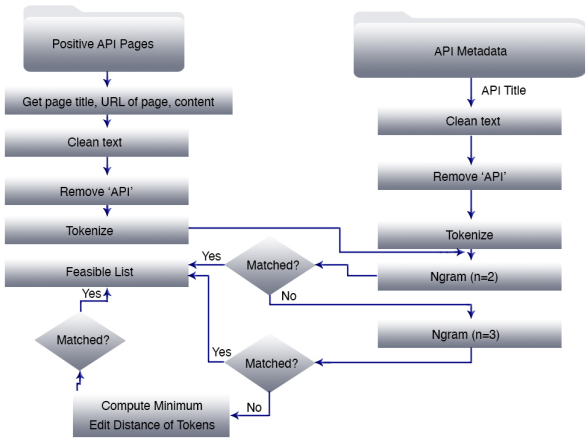


Figure 4.7: Similarity Measurement

accuracy of the mapping process, we also use Minimum Edit Distance as another measurement for our sequence matching process.

API Document restructuring Figure 4.8 explains how the results of API scoping are used. HTML files should be re-organized after their scope is identified. There are three possible scenarios after API scoping of HTML files:

- The new scope shows that the HTML file must be moved from APIx's folder to APIy's folder,
- API scope hasn't changed - no need to change the folder,
- The HTML file has to be removed since the method couldn't find any suitable scope for that.

The role of this component is to organize HTML files containing the API docu-

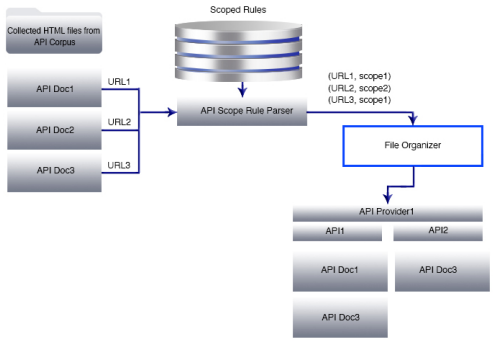


Figure 4.8: API Document restructuring.

mentations into appropriate scopes located in federated folders.

Web-pages collection: In order to implement our parallel Web-crawler, we used Scrapy. Each agent of our crawler explores a web-page pointed by the pointer list and adds embedded links of the page to the list. Once an agent becomes available, it picks one of the pending URLs, collects the web-page content, and updates crawler's pending list.

Table 1 shows the size of the API Corpus that we used in our different experiments. We used different parameters for crawling and collecting web-pages. For example, in the Exp17, we chose MaxDepth8 to be 5 and MaxPage to be 1,000 which resulted in more pages than Exp35 where we chose MaxDepth to be 3 and MaxPage to be 300.

Although using less restrictions in some experiments resulted in collecting a large number of collected web-pages - i.e. experiment 17 - the relevance of collected Web-pages plays a key role in data acquisition. REST API Filter enables us to

pick an API corpus with a greater number of web-pages that are relevant to REST APIs. For instance, experiment 17 collects 2,822,997 web-pages with the total size of 208.6GB and for more than 20,000 public APIs. However, applying REST API filter leaves us with just 5,320 APIs!

4.5 Experiment Results

4.5.1 API Corpus

A common precursor to information extraction from HTML pages is the process of removing unwanted the HTML tags and and producing a machine readable and parsable format like JSON. We cleaned the corpus by i) removing tags, and javascript code from extracted text of HTML web-pages, ii) removing stop words, and iii) stemming and lemmatization the words. We developed a parallel web-crawler for data collection which targets more than 20,000 APIs. About 61% of requests (9,840 requests) have successfully received data from hosts. The major request errors indicate that 21% of requests are *bad request*, 8% are *Not Found* and 6% of requests are *Found (indirect access)*. Those errors can be caused by different reasons, for instance, the error might be raised because the API provider's server is down, the web-site pages is updated and previous links are not available any more, or the API publisher is not in business any more. Our records indicate that around 39% of public API can be terminated which might be the main problem of integrating public APIs in software applications. The major error represents *DNS Lookup Error* with 1,680 requests that indicate the whole DNS of an API publisher

Table 4.1: The size of API Corpus in different experiments.

Exp.#	*MaxDepth*	*MaxPage*	# of files	Total Size (GB)
17	5	1,000	2,822,997	208.6
20	4	100	148,479	8.3
35	3	300	156,497	7.4
37	4	300	256,583	15.0

cannot be reached. We collected more than 208 GB HTML API documentations as shown in 4.1.

4.6 Chapter Summary

We developed an automation tool where a user is able to efficiently annotate REST API documentations and irrelevant API documentations. The annotation tools allow a users to annotate more than 100 APIs with different pages in one day. The annotated data has been used to train the API classifier. Although the performance of API classifier was more than 80% after fine-tuning but scalability of the API classifier in a read-world scenario is much important than performance on k-fold cross-validation on testing data set. We define top providers where the providers offer more APIs. For instance, Microsoft offers a large number of APIs to software engineers. We utilized Scrapy to follow links with a depth of 5 and selected the top 60 providers. In our experiment, we collected 89,740 HTML

Status	200	301	302	303	401	403
of APIs	**1,222**	33	45	2	12	45
Status	405	503	Exception			
of APIs	20	2	97			

Table 4.2: Validation of API Classifier for 1,521 discovered APIs

pages for a total of 1,521 APIs. We designed classifiers to detect positive and negative pages as explained in Section 4.3. Positive pages correspond to html pages, which explain REST API documentations, and negative pages correspond to irrelevant REST API documentations. Our classifier detected 43,356 pages as positive and 39,949 pages as negative, which corresponds to 52.38% and 47.62% of the total pages, respectively. After the API classifier categorized each html page, we extracted the API endpoints from the content of each html page by using transformation-based learning and regular expression learning, which are explained in [Bahrami and Chen, 2020] and [Bahrami et al., 2018]. We submitted a request to extract and validate each API endpoint. Table 4.2 shows the http response and code status response for all requests. As shown in this table, we have received 1,222 successful API requests for a majority of the discovered APIs (out of 1,521), which indicates an 80% performance rate for the API classifiers in a real-world scenario.

Chapter 5

A Deep Signature-based API Specification Learning Approach

Chapter Overview[1]

The number and variety of Web APIs is growing exponentially. Software engineers need to expend a significant amount of time and effort reading and understanding the accompanying documentation. In addition, system automation may use API to interact with each other. However, this is not always a simple task since the API documentation of a provider can be anything from a single HTML page description through to a complex structure with information spanning several pages. Understanding this wide variety of API documentation structures and styles is therefore a labor intensive and error-prone task for engineers. By providing a

[1]This chapter is based on the previously published article: Bahrami and Assefi, et al. "Deep SAS: A Deep Signature-based API Specification Learning Approach." 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020.

machine-learning platform that can extract and standardize API usage information, however, we believe we can accelerate the creation of API-enabled systems by using automation to simplify the task of understanding. In this chapter we introduce a novel approach to automating and standardizing usage information about APIs, combining several machine-learning algorithms in order to extract key attributes from API documentation and generate a machine readable Open API Specification (OAS). We develop i) a content-based learning model that identifies the context of a block of extracted API features; ii) a signature-based machine learning model that recognizes a sequence of successful/unsuccessful extracted API endpoints; and iii) a deep mapping model that pinpoints fine-grained mapping of extracted API attributes to OAS objects. Results of our experiments show that the proposed approach successfully works with an accuracy of 99%, 94% and 97% for content-based learning, signature-based learning, and Deep Mapping of API attributes respectively. We then use the models to produce OAS compliant API Specifications for more than 2,585 public APIs, validate them via API calls and finally deploy the validated APIs to the RunMyProcess software automation platform.

5.1 Introduction

Open API Specification (OAS),[2] allows programmers to outline an entire API by describing the available endpoints, the operations which can be performed on

[2]A sample OAS file and main schema is available at: https://swagger.io/docs/specification/basic-structure/which is also known as *Swagger Specification*

each endpoint and the parameters necessary to perform each operation (plus other related information such as authentications, etc.) The OpenAPI specification therefore enables APIs to be self-descriptive. The objective of this study is to boost the productivity of software engineers by replacing labor-intensive and error-prone API documentation with Open API specifications which are suitable for automated discovery and development. To do this we automatically extract Open API Specifications from a wide variety of API documentation styles with our machine learning methods.

5.1.1 Contribution

The main challenge of our study is to extract API features such as endpoints from heterogeneous API information before mapping them to OAS compliant formats, providing a unified model for API specifications. For example, if we read the API documentation from even the most popular API providers, there is inconsistency in both the terminology and structure of the API documentation. We applied advanced machine-learning algorithms to tackle the variety of styles, formats and terminology found in API documentation and to generate a unified API specification for each API. This chapter introduces several substantial new developments in the mining of API documentation: (i) Developing a content-based learning model to predict the type of the context of each block of API documentation; (ii) developing a signature-based learning model to process metadata of an API description by utilizing the heading HTML tags as a unique signature to recognize an individual OAS object(iii) developing a Deep Mapping Model to specify a fine-grained

OAS object type; (iv) performing extensive experiments to validate the trained models; (v) implementing the proposed components in real-world scenarios and measuring the performance of models on a variety of extracted API features; (vi) collecting a significant amount of API documentation, using the trained models to generate thousands of OAS-based APIs, validating OAS files by performing API calls and finally deploying these validated APIs into our cloud-based visual programming platform.

To the best of our knowledge, our work is the first to apply signature based learning and deep content mapping to generate open API specifications.

5.2 Learning Approach

The majority of API providers use HTML table tags to explain the key information of APIs, such as *parameters* and *responses* in OAS. Our platform extracts all HTML table tags and other important information through a regular expression learning approach which is explained in [Bahrami et al., 2018] and [Bahrami and Chen, 2020]. In this chapter, We proposed three different approaches to predict the type of the extracted API features:

5.2.1 Content-based learning

This module aims to train a model which recognizes different types of API objects. The model extracts API features from HTML table tags. This component uses a ground-truth dataset to detect different OAS API attributes. For instance,

Figure 5.1 shows two HTML table tags with transparent borders. The first table contains the API parameter description while the second table explains the API responses. Content-based learning aims to predicting a type for each extracted HTML table from API documentation. For example, it targets the *parameter* class rather than a fine-grained type such as *parameter name* or *parameter type*. To formulate the problem, let $C = \{c_1, c_2, ..., c_n\}$ be a set of contexts of extracted document and $\tau = \{t_1, t_2, ..., t_n\}$ represents their associated types. The content-based learning task is assigning a new context of c_m to t_k where $k \in [1, .., n]$. We produce a set of classifiers (ϕ) to maximize $P(c_m|\tau, \alpha)$ such that $\forall t_n \in \tau$ where α_i is the parameter of ϕ_i. Therefore, we can estimate the probability of the given context as follows:

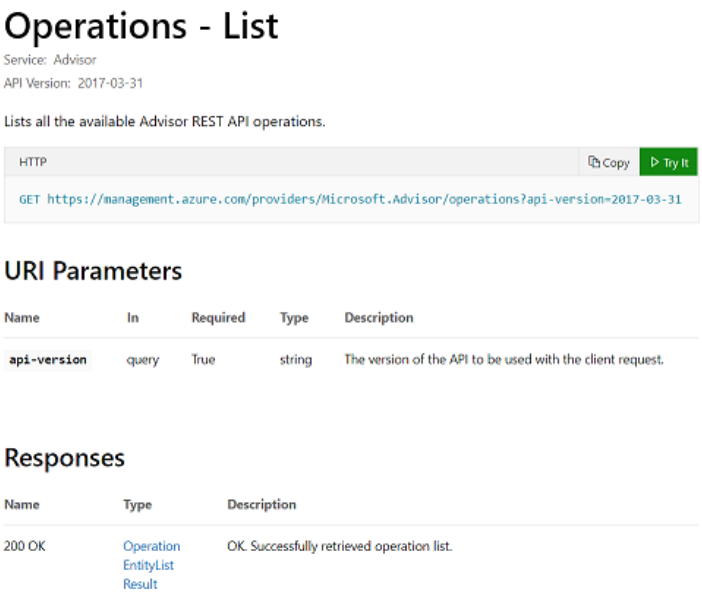
$$P(c_m|\tau) = \frac{\sum_1^{|\tau|} P(c_m|t_n)}{|\tau|} \tag{5.1}$$

Each c_m contains a set of tokens (φ) where each one includes a word from the API corpus dictionary[3]. By training our model, we will be able to predict and assign c_m to t_n according to the output of $maximize\ P(c_m|\tau, \alpha)$.

5.2.2 Signature-based learning

When we process the context of a table, we might still have an issue caused by the incorrect classification of a HTML table into an incorrect API attribute. The incorrect classification may cause an invalid OAS API mapping in addition to an incorrect API specification. Any issues in API specification may cause connection failure to the API server. In order to detect False-Positive outputs from the

[3]The list of all used tokens from all API documentations.



Operations - List

Service: Advisor

API Version: 2017-03-31

Lists all the available Advisor REST API operations.

HTTP Copy Try It

GET https://management.azure.com/providers/Microsoft.Advisor/operations?api-version=2017-03-31

URI Parameters

Name	In	Required	Type	Description
api-version	query	True	string	The version of the API to be used with the client request.

Responses

Name	Type	Description
200 OK	Operation EntityList Result	OK. Successfully retrieved operation list.

Figure 5.1: Sample API Documentation of Azure Advisor API.

content-based learning, we develop a model that recognizes different API features according to their headers. For instance, the HTML tables in Figure 5.1[4] have a set of headers and this additional metadata can help with both API feature extraction and OAS mapping. We develop a model that identifies true/false positive results of content-based learning model by processing their previous HTML heading tags. Consider Figure 5.1 as an example! If the content-based learning model could not recognize the type of a HTML table tag - *responses* and *parameters* in this case - then the set of previous HTML headers can help to identify the correct type. For instance, *"Operations-List"* in Figure 5.1 indicates that an endpoint object is

[4]The complete page can be retrieved from "https://docs.microsoft.com/en-us/rest/api/advisor/operations/list".

coming next; *"URI Parameters"* indicates that a *Parameter* context is going to appear; and *"Responses"* indicates a context associated with *Responses* object in OAS. I also shows two attributes of the endpoint(i.e. parameter and response). Intuitively, the chain of all previous headers of each block in the document allows us to train a model to recognize different API specifications. For example, Figure 5.2 shows a snapshot of Google Abusive API documentation[5] on the right and the chain of HTML headers tags that appear before the endpoints on the left. Two endpoints are highlighted in the figure. We developed a regular-expression model to extract API endpoints (i.e., *GET /farm/v1/animals/pony*). Although *GET /farm/v1/animals/pony* can be detected as an endpoint by content-based learning, however, the chain of headers indicates that the extracted endpoint is a false-positive endpoint because previous headers of the endpoint show that it is an example value ("Example batch request"). The example values should not be added into OAS. To formulate our signature-based learning model, let $\mathcal{S}$ be the signature of Θ, and Θ is an entity which is extracted from the OAS object:

$$\mathcal{S}_{\Theta} = \{\aleph, h_1, h_2, ..., h_n\} \tag{5.2}$$

$$h_i^{i=1,...,n} \in \{H1, H2, ..., H6\} \tag{5.3}$$

($\aleph$) denotes the HTML page title of the extracted content, and $h_1, h_2, ..., h_n$ denotes its associated sequence of HTML heading tags. Let $\Theta \in \{EP, Para, Resp, Sec, SecDef\}$

[5]Complete page can be retrieved from https://developers.google.com/abusive-experience-report/v1/how-tos/batch

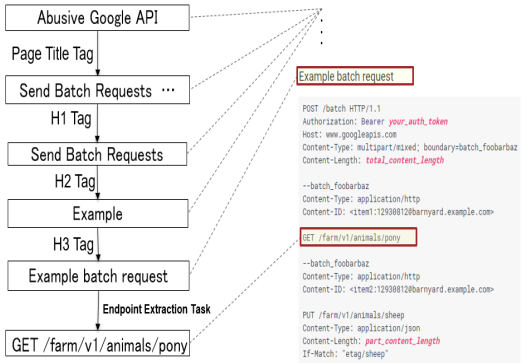


Figure 5.2: Sample Header chain of signature-based learning for Google Abusive API.

where EP, $Para$, $Resp$, Sec, and $SecDef$ represent *Endpoint*, *Parameter*, *Response*, *Security*, and *Security Definition* in the OAS, respectively. Each header (h_i) represents a HTML heading tag type and $h_i \in \{H1, H2, ..., H6\}$ according to W3C recommendations [Consortium et al., 1999].

Endpoints' Headers Chain Construction

Let $\mathcal{A}_\mathcal{S}$ be extracted information from an API documentation dataset ($IE_{Dataset}$). In addition to endpoints, $\mathcal{A}_\mathcal{S}$ contains the page title and the chain of HTML heading tags (headers) that appear before the extracted endpoint.

Algorithm 5 shows the process of generating a chain of headers from the endpoints extracted from API documentation $\mathcal{A}_\mathcal{S}$ in comparison with OAS ground-truth dataset ($GT_{Dataset}$), if it is matched with an endpoint of $\mathcal{A}_\mathcal{D}$; otherwise it is added as a $False$ value in the $GT_{Dataset}$. The output of Algorithm 5 is $\mathcal{S}\theta_i$

for ith API document of θ API. We use $GT_{Dataset}$ to train a classifier to predict true positive endpoints according to the extracted page title and headers, which corresponds to signature of each OAS object. One of advantages of Signature-based learning is that the model is able to learn false-positive endpoints from the extracted information as shown in Figure 5.2 according to its $\aleph$, and $h_1, h_2, ..., h_n$.

Algorithm 5 Producing a chain of headers for Endpoints

procedure ENDPOINT_HEADER_CHAIN()
▷ Add headers ($\mathcal{H}$), extracted Endpoint (EP) of extracted information Dataset ($IE_{Dataset}$) as True, if it founds in Available OAS Files ($Avail_{Dataset}$).
    for $\mathcal{A}_\mathcal{D}$ *in* $IE_{Dataset}$ **do**
        for $< EP_\mathcal{D}, HT_\mathcal{D} >$ *in* $\mathcal{A}_\mathcal{D}$ **do**
            $Found \leftarrow False$
            for $\mathcal{A}_\mathcal{S}$ *in* $Avail_{Dataset}$ **do**
                if $< EP_\mathcal{D}, HT_\mathcal{D} >$ *in* $\mathcal{A}_\mathcal{S}$ **then**
                $\mathcal{S}\theta_i \leftarrow < EP_\mathcal{D}, HT_\mathcal{D}, \aleph_\mathcal{D}, \mathcal{H}_\mathcal{D}, True >$
                $Found \leftarrow True$
                end if
            end for
            if *Not Found* **then**:
                $\mathcal{S}\theta_i \leftarrow < EP_\mathcal{D}, HT_\mathcal{D}, \aleph_\mathcal{D}, \mathcal{H}_\mathcal{D}, False >$
            end if
        end for
    end for
end procedure

Another advantage of defining the header chain for the endpoint is predicting *RESTful HTTP verb* functions, such as $POST, PUSH$ and $DELETE$ by a given signature ($\mathcal{S}\theta_i$) of $EP_\mathcal{D}$.

Tables' Header Chain Construction.

Similar to the endpoints' header chain construction, we need to produce a ground-truth of headers for each extracted HTML table tag. Let $\mathcal{A}_{\mathcal{D}_T}$ denotes extracted HTML table from API documentation ($\mathcal{A}_{\mathcal{D}}$). However, the content of an HTML table might not be exactly matched to an API specification. Therefore, $\mathfrak{T}(\mathcal{A}_{\mathcal{S}_T}) \subseteq \mathfrak{T}(\mathcal{A}_{\mathcal{D}_T})$ where $\mathfrak{T}(.)$ denotes a function that produces the list of tokens of a given OAS value because regularly $\mathcal{A}_{\mathcal{S}_T}$ contains a summary of $\mathcal{A}_{\mathcal{D}_T}$, which includes more detail and descriptions. For instance, the documentation of an API may explain the detail of an *error response*, but a summary of the *error response* can be provided in OAS file. We use an algorithm to detect a partial match between $\mathcal{A}_{\mathcal{D}_T}$ and $\mathcal{A}_{\mathcal{S}_T}$. We define a normalized ratio partial match of $\mathcal{A}_{\mathcal{S}_T}$ to $\mathcal{A}_{\mathcal{S}_T}$ (i.e., 80% of $\mathcal{A}_{\mathcal{S}_T}$) in Algorithm 6. A ratio is defined as $Table\_Match()$ procedure and it produces tokens (Γ) of $\mathcal{A}_{\mathcal{D}_T}$ and $\mathcal{A}_{\mathcal{S}_T}$ to find the matching ratio and returns *True* if it is greater than a matching threshold (λ).

5.2.3 Deep Content Mapping

As we explained in the previous sections, the model is trained to identify a general type of OAS object by processing the whole context of extracted API information. For example, content-based learning and signature-based learning may recognizes the content of HTML table as a *Parameter* object of the OAS. Now, we need to map each column of the extracted HTML table to a specific OAS sub-object. For instance, it may map the first column of an extracted HTML table to *Parameter Name* and the second column of the table to *Parameter Description*.

Algorithm 6 Producing a chain of headers for HTML Tables

1: **procedure** TABLE_HEADER_CHAIN()
▷ Add headers of extracted Table ($\mathcal{A}_{\mathcal{D}}$) in ($\mathcal{S}_{\theta}$) as True, if there is a content matched with an object of $\aleph_{\mathcal{D}}$ in ($Avail_{Dataset}$).
2: **for** $\mathcal{A}_{\mathcal{D}_T}$ *in* $IE_{Dataset}$ **do**
3: **for** $\mathcal{A}_{\mathcal{S}_T}$ *in* $Avail_{Dataset}[\aleph]$ **do**
4: $Matched \leftarrow False$
5: **if** $Table\_Match(\mathcal{A}_{\mathcal{D}_T}, \mathcal{A}_{\mathcal{S}_T})$ **then**
6: $\mathcal{S}\theta_j \leftarrow < \mathcal{A}_{\mathcal{D}}, HT_{\mathcal{D}}, \aleph_{\mathcal{D}}, \mathcal{H}_{\mathcal{D}}, True >$
7: $Matched \leftarrow True$
8: **end if**
9: **end for**
10: **if** $Not\ Found$ **then**:
11: $\mathcal{S}_{\theta_j} \leftarrow < \mathcal{A}_{\mathcal{D}}, HT_{\mathcal{D}}, \aleph_{\mathcal{D}}, \mathcal{H}_{\mathcal{D}}, False >$
12: **end if**
13: **end for**
14: **end procedure**
15: **procedure** $Table\_Match(\mathcal{A}_{\mathcal{D}_T}, \mathcal{A}_{\mathcal{S}_T})$
$\Gamma_k \leftarrow Tokens(\mathcal{A}_{\mathcal{S}_T})$
$\Gamma_m \leftarrow Tokens(\mathcal{A}_{\mathcal{D}_T})$
16: $\mathcal{C} \leftarrow (\Gamma_k \cap \Gamma_m)/Len(\Gamma_k)$
17: **if** $(\mathcal{C} > \lambda)$ **then**
18: $Return\ True$
19: **else**
20: $Return\ False$
21: **end if**
22: **end procedure**

Fine-grained mapping is more challenging because a wide variety of API documentation styles exist. In addition, there might be some overlap between columns that may cause confusion for the classification model. We develop a character embedding CNN [Kim et al., 2016] ,which uses BiLSTM [Ding et al., 2018] model to predict the mapping of each column of a HTML table to a fine-grained unified API specification attribute. By processing the character level of a value of an OAS object, the deep mapping model can find different patterns between different OAS objects. Our goal is to find character based patterns between different API attributes on top of sentence mining and document mining of an API. The architecture of the character embedding CNN for the deep mapping model is shown in Figure 5.3. Let $\mathcal{M}$ denote a character Embedding CNN model as introduced by Zhang et al. [Zhang and LeCun, 2015]. $\mathcal{M}$ takes all OAS files as input and classifies values of an object per key in JSON OAS. We define $\mathcal{M}_\zeta$ where $\zeta \in \{1, 2, ..., \eta\}$ that represents mapping of each OAS object. Therefore, ζ denotes all OAS object types (generic OAS attribute names), which is classified by the content-based and the signature-based learning models as described in Section 5.2.1 and 5.2.2, respectively. Defining only one deep mapping model may not provide a good performance due to the large number of sub-objects in OAS. However, by defining η number of models, we expect $\mathcal{M}_\zeta$ to provide a better performance over classifying all fine-grained OAS object types (all API attributes) at the same time. For instance, let $\mathcal{M}_1$ ($\zeta = 1$) represent *Parameter* OAS object type; once the context of a table is detected as a *Parameter* type in the content-based and signature-based learning models, then $\mathcal{M}_1$ detects each single object of the *Parameter* type of the

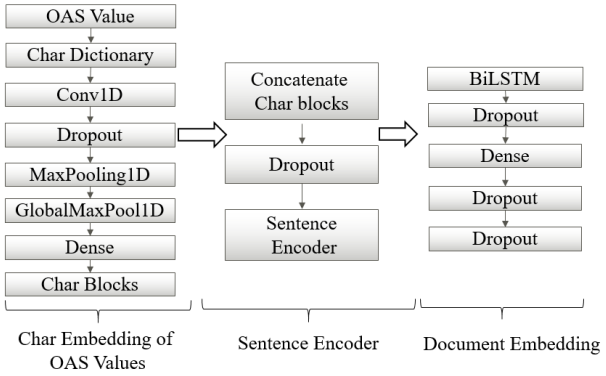


Figure 5.3: The overview architecture of Char-embedding model of deep OAS mapping.

OAS. *Parameter* types of the OAS include *Parameter Name*, *Parameter Description*, *Parameter Default Value*, *Parameter Maximum value*, and other sub-objects.

5.3 Experiments

We extract API information from documentation via the proposed DeepSAS for a large number of APIs. We also collect available OAS files from API publishers and third-parties through a web-crawler and use them as the ground-truth dataset for training our models. In these experiments, we use precision, recall, and f1-Score and accuracy [Pedregosa et al., 2011b] for evaluation of our trained models. These techniques are standard evaluation methods for classification problems.

5.3.1 Data Acquisition

We use several sources, such as ProgrammableWeb, API Guru, API Harmony, RapidAPI, API Mashup etc., to collect a comprehensive list of APIs, and use the crawler to collect API documentation for more than 20,000 APIs. In our study, we target only REST APIs.

We use a web-crawler composed of 32 parallel web-crawler agents. Each web crawler agent collects partial API documentation, and the collection of all API documentation generates our $APICorpus$.

We collected $2,822,997$ web-pages with a combined size of $208.6GB$ for more than 20,000 public APIs. However, not all of collected public APIs are categorized as REST API; therefore we used a *REST API Filter* [Bahrami et al., 2018] that identified 5,320 APIs as a REST API group. We clean the API corpus by i) removing scripts from extracted text of HTML web-pages, ii) removing stop words, and iii) stemming the words.

5.3.2 Content-based Learning Evaluation

We use Scikit-Learn [Pedregosa et al., 2011a] in Python to train different models as described in the previous section. In order to process the content of JSON files (OAS-based API specifications), we produce a set of classes, τ where each class corresponds to a type of OAS object (JSON Key) in JSON file. Then, we produce the context of each key (a string value of each key), C as described in Section 5.2.1. We use Tf–idf term weighting [Joachims, 1996] in Scikit-Learn to extract features for the contexts (C). We used 75% of the dataset for training and the

remaining 25% for testing.

Datasets

In order to train different machine-learning algorithms, we use all available OAS-based API Specifications, totalling 1,726 JSON files. In this dataset, 112 files of 1,726 files do not contain any endpoint information (*Path Object* in OAS). We produce a set of strings for each type where each JSON key is considered as the type of object (τ) and the string (value) is considered as context (C) in Equation 5.1. The value of a JSON key corresponds to the actual table context in API documentation HTML files. Our dataset contains 45,977 sample data where 29,884 have been selected for training and 16,093 for testing. Our sparse vectorizer contains 13,299 records which produces a dataset with shape (45977,13299) as the training dataset and a testing dataset with shape (16093,13299). We consider several key objects from OAS that includes ***info, endpoint, responses, parameters, basePath, security Definitions, schemes, consume/produces*** and ***security*** OAS objects [6].

Evaluations

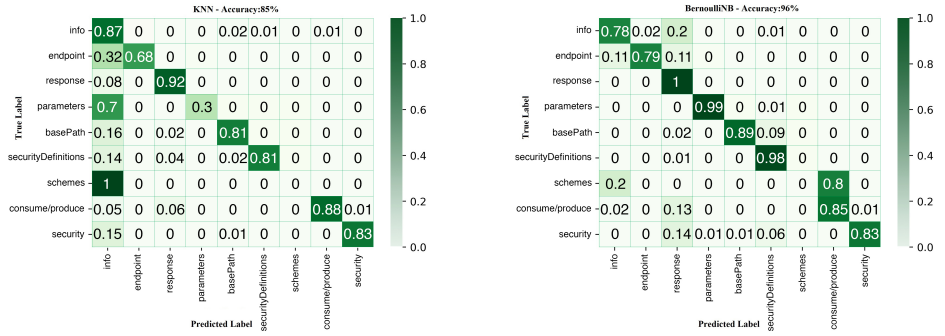
We use 16 different classifiers to validate the testing dataset and evaluate each model. Figure 5.4 shows normalized confusion matrices of validation dataset for only three different classifier algorithms including K-Neighbors Classifier (Fig. 5.4a), Naïve Bayes classifier (Fig. 5.4b), and LinearSVC (5.4c). K-Neighbors

[6]OAS schema is available at https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md Retrieved on Jan 27, 2019

Classifier detects 9 different OAS objects with an accuracy of 85% where the majority of incorrect classifications happen in detecting *info object* and *schemes* object of OAS. *Info Object* contains a variety of unique words due to considering *API title* as a sub-object of *Info Object*. The second algorithm is a Naïve Bayes classifier for multivariate Bernoulli model [Schütze et al., 2008] which is more practical for discrete data as well as a binary classification [Schütze et al., 2008]. As shown in the figure 5.4b, it detects objects with an accuracy of 96%. Finally, LinearSVC performs as the best algorithm in our dataset where it detects OAS objects with an accuracy of 99%. The detail of LinearSVC validation is explained in Table 5.1 which includes a total 16,093 of sample data in the testing dataset. We use LinearSVC model as content-based learning model to predict a sheer number of HTML table context of *API Corpus*. Usage of the model is explained in Section 4.

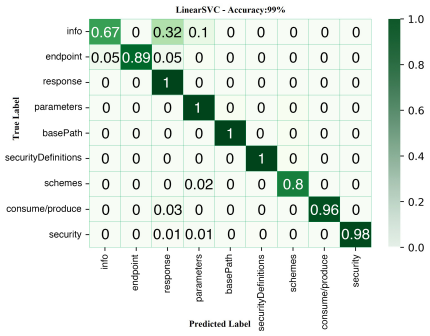
Table 5.1: Evaluation results for OAS content-based learning for LinearSVC with configuration of $loss = squared\_hinge$, $max\_iter = 1000$, $penalty = l2$, $tol = 0.001$.

OAS Object	**precision**	**recall**	**f1-score**
basePath	0.97	0.67	0.79
consume/produces	1	0.89	0.94
endpoint	0.99	1	1
info	0.99	1	0.99
parameters	1	1	1
responses	1	1	1
schemes	1	0.8	0.89
security	1	0.96	0.98
security Definitions	0.99	0.98	0.98
avg / total	0.99	0.99	0.99



(a) k-Nearest Neighbors Model Validation.

(b) Multivariate Bernoulli Model Validation.

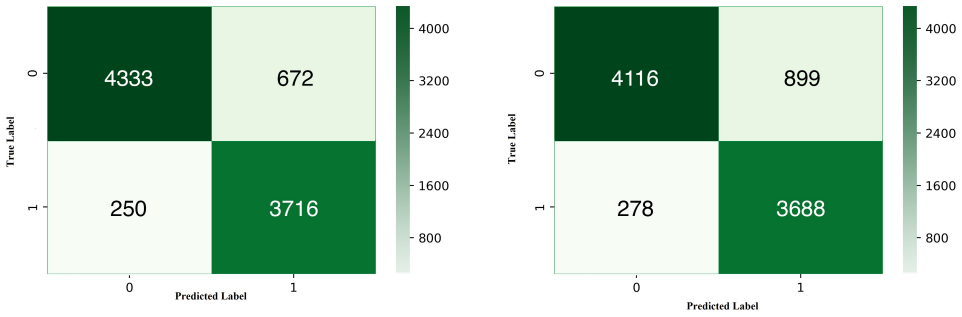


(c) LinearSVC Model validation.

Figure 5.4: Confusion matrix of content-based learning model for predicting table contexts.

5.3.3 Signature-based Learning Evaluation

We need a ground-truth dataset for both endpoints and HTML table tags because we do not have heading HTML tags and page titles for each object in the available OAS file repository. This section explains the construction of the ground-truth and the performance of prediction models on the validation dataset. Firstly, we extract information through our API learning platform [Bahrami and Chen,



(a) (b)

Figure 5.5: Confusion matrix for endpoints classification.

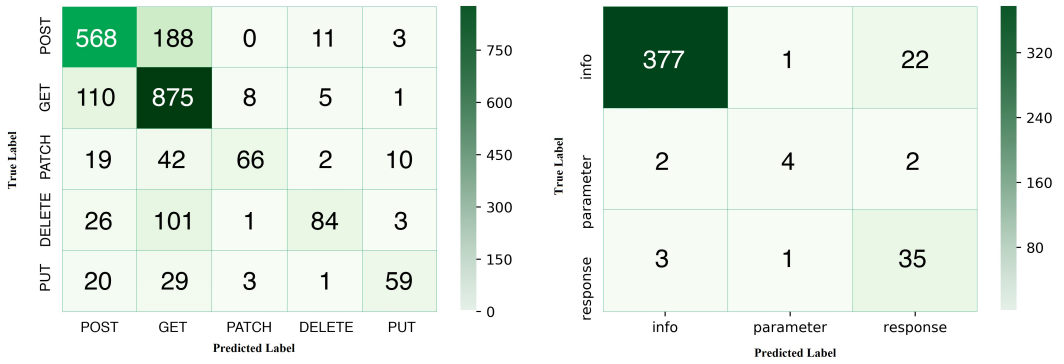
2020][Bahrami et al., 2018], then we check if the extracted information appears in the available OAS repository as explained in Section 5.2.2.

Endpoint Processing

Algorithm 5 shows the process of labeling every potential endpoint from API documentation as true or false.

After labeling the dataset by using Algorithm 5, the next is preparing the training and validation datasets. We chose to include HTML headers, HTTP verb functions, and OAS object type in the training dataset. Different sets of features then can be used to train different machine learning models in order to pick the most promising.

For the first attempt, we consider a combination of all headers as a single content. The dataset consists of 5,005 sample data with the *False* value, indicating that the associated endpoint did not appear in $Avail_{Dataset}$ (matched endpoint),



(a) (b)

Figure 5.6: Confusion matrix: a) Predicting HTTP verb functions based on signatures, b) Table type prediction.

and 3,996 sample data with the *True* value, indicating that the associated endpoint did appear in $Avail_{Dataset}$. Since all of the endpoints extracted from the API documentation set are considered as *Positive* extraction by training the model, it identifies *True Positive* or *False Positive* endpoints according to the signature. A Multinomial Naïve Bayes classifier of Scikit-learn is used for this binary classification task. Figures 5.5(a) and Table 5.2 show the results of the classification for the dependent variable. To ensure that the models do not have an over-fitting issue, we also perform prediction on several APIs (as a cross-validation) where the model successfully predicted APIs with an accuracy of 92%. Each header item in $\mathcal{S}_\Theta$ is considered a relevant factor. The model considers the order of headers as they appeared in documentation as $\aleph \rightarrow h_1 \rightarrow h_2 \rightarrow ... \rightarrow h_n$.

As another attempt, we also train the model by considering each HTML header

Table 5.2: Evaluation Results.

	precision	**recall**	**f1-score**
2.1 Endpoints Classification			
Dependent Variable			
False	0.95	0.90	0.92
True	0.93	0.96	0.94
avg / total	0.94	0.94	**0.94**
Independent Variable			
False	0.95	0.86	0.90
True	0.84	0.94	0.89
avg / total	0.90	0.90	0.90
2.2 Predicting the HTTP Functions			
POST	0.76	0.74	0.75
GET	0.71	0.88	0.78
PATCH	0.85	0.47	0.61
DELETE	0.82	0.39	0.53
PUT	0.78	0.53	0.63
avg / total	0.75	0.74	0.73
2.3 Table Processing			
False	0.94	0.99	0.96
Parameter	0.50	0.67	0.57
Response	0.93	0.93	0.93

in the signature as a separate variable. The performance of this model is shown in Table 5.2 and Figure 5.5(b). In addition we designed another classifier for predicting HTTP verb function according to its signature. The dataset contains two columns: the first containing all headers concatenated and the second containing the http function. This model predicts a HTTP verb function if it is missing in the API documentation. The model predicts *POST, GET, PATCH, DELETE* and *PUT* with an average f1-score of 73%. This performance means our model can

predict HTTP verb of an endpoint even without appearing in API documentation. The results are reflected in Table 2.2 and Figure 5.6a.

Processing the tables

We consider all Google API documentation and associated OAS files. In order to extract and label table contents, we use Algorithm 6. Each HTML table object extracted from the API documentation is compared against *Avail* and labeled as *True* or *False* accordingly. Once prepared we then use the extracted labels to train a multinomial Naïve Bayes classifier model using Scikit-learn. The objective of this model is to predict the type of table according to its signature. Due to restricted matching criteria ($\lambda = 0.4$), we found 447 samples that categorized as one of three classes: *False*, *Parameters* and *Responses*. In future work, we plan to develop a new approach to improve the labeling process and to apply the method on a larger dataset. Figure 5.6b and table 2.3 show the classification result for table processing.

5.3.4 Deep Mapping Evaluation

The goal of this model is to map an HTML tag to a specific OAS object type. There are some similarities between objects in each OAS object type. For instance, *Parameter Name* and *Parameter Type* may appear in *Parameter Description*. As described in the *Deep Content Mapping* section, we produce $\mathcal{M}_\eta$ where $\eta \in [1, .., 5]$, that includes detected objects in *Process content-based learning* section which consists of *Info*, *Responses*, *Parameters*, *Security Definitions* and *Security*,

respectively. We do not consider some other object types in our content-based and signature-based learning models, such as *endpoint* and *basePath*, because those objects are already a fine-grained OAS type. In this section, we focus on two major OAS objects and their performance including *Parameter* and *Info*. We use Keras [7] and Tensorflow [8] as the back-end technologies for model development.

Info Object. This type consists of *API Description, API Title, Terms of Services, which includes the URL and a term title*, and *API version*. The character embedding includes 93 different characters in the dictionary. We consider a maximum number of 5 sentences per OAS value with a maximum length of 256 characters. As described, the OAS contains a summary of API description; as a result, only 28 of the values reach the maximum length for the sentences. The shape of the training dataset is (4386, 5, 256) and the testing dataset is (1097, 5, 256). The accuracy of $\mathcal{M}_{Info}$ on validation dataset is 88.61%.

Parameter Object. Parameter object type consists of parameter info such as *name* and *type*. Similarly to *info* object, character embedding includes the maximum of the first 93 different characters of each token. We consider a maximum number of 5 sentences per OAS value, with a maximum length of 256 characters. 107 OAS values reach the maximum length of sentences. The training dataset dimension is a matrix of (304691, 5, 256) and the testing dataset is (76173, 5, 256). The accuracy of $\mathcal{M}_{Parameter}$ on validation dataset for 19 different fine-grained OAS objects (API attributes) of *Parameter* is 97.5%. Compared to $\mathcal{M}_{Info}$, the content

[7]Keras is a Python Deep Learning library and it is available at: https://keras.io

[8]TensorFlow is an open-source machine learning library and it is available at: https://www.tensorflow.org

of each *Parameter* type has less overlap with each other (less confusion as shown in Figure 5.5. Therefore $\mathcal{M}_{Parameter}$ shows a better performance over $\mathcal{M}_{Info}$.

5.4 Deployment

We believe that the increasing availability of web-connected business and technology services will accelerate the shift towards new value-creation models in which people simply link together existing assets and resources. Our ultimate aim is facilitating such business model creation and experimentation by making the implementation of a new business model as straightforward as sketching the idea and connecting it to the resources needed to realize it. Our suite of API learning technologies and Deep SAS, which aim to generate a unified repository of usable API specifications from heterogeneous sources, offers one of the key foundational pillars of such an environment by providing the necessary 'building blocks'. By generating a repository of Open API Specifications and associated metadata our ecosystem building component creates an automatically growing source of services which can be used in the creation of higher order systems. In order to turn an idea into working businesses, we need to be able to move beyond imagination and extend our environment into business model design and strategy. We fulfill this need by highly customizing Wardley Mapping [Wardley]. In this technique, people focus on first understanding user needs before mapping out the value chain of components necessary to realize all requirements. We use models to process all collected API documentation. We produced OAS-based API Specifications for

more than 2,585 public APIs, where each API has at least an endpoints-attribute (e.g., *Parameters* and/or *Responses* and etc). We validate APIs by calling each API endpoint. The API validation component allows us to understand if an extracted endpoint is valid or not, helping us to successfully validate 59,052 API endpoints. To use these endpoints in the creation of applications we have tested the use of a cloud-based software implementation platform called RunMyProcess (RMP) [Daniels et al., 2013] [9] for deployment of the output of the DeepSAS. RMP uses two objects to fully describe an integration endpoint i) the API provider (or system) and ii) the API connector (or specific endpoint within a system). To enable the seamless use of the outputs of our proposed approach, we developed a component to automatically generate API providers and connectors in RMP using validated APIs. We use OAS compliant APIs to produce i) API provider which represents the API Provider definition (*host* in OAS), and ii) an API connector that represents API endpoints in RMP. When creating a software application in RMP a developer can model e.g. web applications or business processes which in turn can utilize connectors to submit or retrieve information from external systems via APIs. Once the generated APIs are added as connectors within the RMP environment, developers are able to freely use the generated API connectors within their software project designs. Figure 5.7 shows an automate generated API provider and the list of endpoints (API Connectors) in RMP. In addition to this development acceleration we are also experimenting with simple use-cases which test the potential for business and IT alignment during value chain mapping.

[9]RunMyProcess platform is available at: https://www.runmyprocess.com

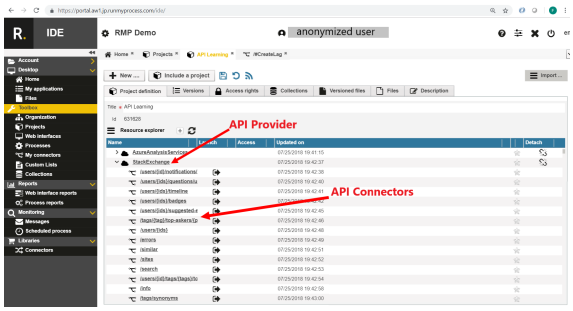


Figure 5.7: A screenshot of added API providers and API connectors in RunMyProcess.

To achieve this we built a minimum viable product of our business modelling environment to model a value chain of the business and map each required business capability to a specific API. The method can be optimized for using a light-weight prediction in deployment [Bahrami et al., 2016].

5.5 Related works

In our previous study [Bahrami et al., 2018] [Bahrami and Chen, 2020], we explained how a variety of information can be extracted from API documentation by utilizing regular expression learning to define the mapping procedure. In this chapter, we focus on mapping API attributes to a unified structure of Open API Specifications by training different machine-learning algorithms. Yang *et al.* developed an automatic approach for crawling API documentation pages and extracting base URL, path templates, and HTTP methods, all together marking out endpoints of an individual API [Yang et al., 2018]. In order to compare our

Table 5.3: A comparison between different platforms

Service Provider	C1	C2	C3	C4	C5 [10]
DeepSAS	Y	Y	N †	Y	2,585
DeepSAS + RMP platform[11]	**Y**	**Y**	**Y**	**Y**	2,585
Programmable Web[12]	Y	N	N	N	22,003
IBM API Harmony[13]	Y	Y	N	N	1,179
API Guru[14]	Y	Y	N	Y	1,552
Anypoint Platform[15]	Y	N	Y	Y	120
Rapid API[16]	Y	N	N‡	N	8,000
Amazon Gateway[17]	N	N	Y	N	-
Amazon AWS Lambda[18]	N	N	Y	N	-
Google Discovery[19]	Y§	Y†	N	N	442¶

C1: Does provide an API Directory (the list of APIs)? C2:Does provide API Specification? C3: Does provide a platform to design Software application based on APIs? C4: Support on-premise APIs? C5: The number of listed APIs (as 7/3/2019).
Y:Yes; N:No; U:Unknown; §: Only Google APIs; †:Only Google proprietary format; ‡:Only Try out; ¶:including old versions

Table 5.4: Comparison of related work

Feature	**DeepSAS**	**D2Spec** (Yang et al. 2018)	**API Learning** (Bahrami et al. 2018)
Host extraction(H)	Yes	Yes	Yes
Base Extraction (B)	Yes	Yes	Yes
Endpoint Extraction (E)	Yes	Yes	Yes
# of HBE[20] evaluation (testing dataset)	**∼431 APIs (25% of 1,726)**	116 APIs	-
F1-score	73%	84.16% (smaller dataset)	-
Parameter Extraction	**Yes**	No	Yes
Parameter Type Detection	**Yes**	No	Yes
f1-score	**99%**	N/A	N/A (81.29% Acc.)
Fined-grain Parameter Detection	Yes	No	No
f1-score	**83.32%**	N/A	N/A
#of processed API documentations	**2,585**	120	1,923
# of labeled APIs	**1,726**	120	200
Endpoint Evaluation Method	HBE Matching, API Call.	HBE Matching	Manual Annotation, API Call
Info Object Detection	88.61%	N/A	N/A
Parameter Object Detection	97.5%	N/A	N/A

experimental results against related work, we consider two approaches for evaluation: **First**, we compare our proposed approach in terms of API coverage and usage. Table 5.3 shows an end-to-end evaluation and comparison. Although Programmable Web has 22,003 APIs listed within its platform it only consists of a directory for metadata ($C1$). Rapid API also relies on user input and does not construct or maintain API specifications in an automated manner. It also does not offer a platform to generate a software application by using its platform. To the best of our knowledge none of the existing platforms are able to cover a similarly large number of APIs which are automatically generated and maintained directly from documentation and easy for developers to integrate within applica-

tions. This evaluation shows that our proposed DeepSAS and RMP platform not only has better API coverage but also allows users to directly access and develop against API specifications ($C2$) by using RMP Connectors ($C3$). In the **Second** approach of evaluation, we compare the performance of our proposed approach against other API documentation processing approaches. The results are shown in Table 5.4. As shown in this table, we not only construct host, base and API endpoint but also produce detailed information of associated parameters and security information, something which D2Spec does not support. Although D2Spec outperforms our approach on F1-score of construction of HBE, it only covers 116 APIs vs our total of 2,585. D2Spec applies classification and hierarchical clustering. It outperforms our model on endpoint extraction only from a small number of APIs. Our proposed approach, on the other hand, applies transformation-learning in the previous work, deep content mapping, signature-based learning and ML classification by using Deep SAS while also being capable of processing a large number of APIs to extract API endpoints, attributes and attribute types. D2Spec suffers from a lack of API attribute extraction and API attribute type detection, when compared to our model.

5.6 Chapter Summary

This chapter introduces a Deep Signature-based API Specification (DeepSAS) construction approach which aims to overcome the challenges of learning from a wide variety of API documentation types. We developed i) a classifier as a

content-based learning component that predicts each extracted API information according to Open API Specification (OAS) object types; ii) a signature-based learning model which predicts OAS object types (API attributes) according to each unique HTML metadata path that appears in API documentation. The metadata includes HTML page title and HTML header tags; and iii) a deep mapping model that maps all fine-grained extracted API specifications to a unified OAS API Specification. DeepSAS uses limited annotation but outperforms on a large dataset of API documentations. Finally we used our models to produce and validate OAS compliant API Specifications for more than 2,585 public APIs and to deploy validated API specifications to the RunMyProcess cloud software platform.

Chapter 6

Conclusion

The amount of digital data being collected continues to expand at an exponential rate, and as a result, the science of data analytics becomes more technologically advanced thereby enabling data scientists to turn this large body of structured and even unstructured data into more reliable information and facts. Computer engineers and scientists have already enlisted the help of big data machine-learning components to solve the problem of pattern discovery from large-scale data sources more efficiently. In this study, we applied machine learning, natural language processing, and big data analytics techniques to collect a sheer number of API documentations and train classifiers to map these API documentations onto a unified model for API specifications. Apache Spark MLlib is a very strong tool for big data analytics that presents spectacular performance in terms of running time and accuracy. Apache Spark MLlib offers fast, flexible, and scalable implementation of a variety of machine learning components, ranging from ensemble learning and

principal component analysis (PCA) to optimization and cluster analysis. Such a tool also presents options for distributed processing and support for big data tools that utilize distributed architectures. Apache Spark played a major role in our ML-based API crawler. We ran an experimental evaluation of Apache Spark to justify its advantages in designing a parallel and distributed web-crawler, which was a crucial component of our study. The results of our experiments, which are reflected in Chapter 3, show that Apache Spark and MlLib have considerable advantages in terms of running time and accuracy.

We developed an automation tool to efficiently annotate REST API documentations, irrelevant API documentations and an API reference page. The annotation tools allow us to label more than 100 APIs with different pages in one day. The annotated data has been used to train the API classifier. Although the performance of our API classifier was above 80% after fine-tuning, utilizing the API classifier in a real-world scenario is much more important than its trial performance. We included APIs from top providers. For instance, Microsoft offers a large number of APIs to software engineers. We utilized Scrapy to follow links with a depth of 5 and selected the top 60 providers. In our experiment, we collected 89,740 HTML pages for a total of 1,521 APIs. We designed classifiers to detect positive and negative pages as explained in Section 4.3. Positive pages correspond to HTML pages, which explain REST API documentations, and negative pages correspond to irrelevant REST API documentations. Our classifier detected 43,356 pages as positive and 39,949 pages as negative, which corresponds to 52.38% and 47.62% of the total pages, respectively. After the API classifier categorized each HTML

page, we extracted the API endpoints from the content of each HTML page by using transformation-based learning and regular expression learning, which are explained in [Bahrami and Chen, 2020] and [Bahrami et al., 2018]. We submitted a request to extract and validate each API endpoint. Table 4.2 shows the http response and code status response for all requests. As shown in this table, we have received 1,222 successful API requests for a majority of the discovered APIs (out of 1,521), which indicates an 80% performance rate for the API classifiers in a real-world scenario. We introduced a Deep Signature-based API Specification construction approach, which aims to overcome the challenges of learning from a wide variety of API documentation types.

We developed

- a classifier as a content-based learning component that predicts each bit of API-extracted information according to Open API Specification (OAS) object type;
- a signature-based learning model, which predicts OAS object type (API attribute) according to each unique HTML metadata path that appears in the API documentation. The metadata includes the HTML page title and header tag; and
- a deep mapping model that maps all fine-grained extracted API specifications to a unified OAS API Specification.

Our models can predict and map objects with an accuracy of 99%, 94% and 97% for content-based learning, signature-based learning, and Deep Mapping of

predicted parameters' detail, respectively. DeepSAS uses limited annotation but outperforms on a large dataset of API documentations. Finally, we used our models to produce and validate OAS-compliant API Specifications for more than 2,585 public APIs and to deploy validated API specifications to the RunMyProcess cloud software platform. We then used the models to produce OAS-compliant API Specifications for more than 2,585 public APIs, validated them via API calls, and finally deployed the validated APIs to the RunMyProcess software automation platform. Application developers and engineers need to spend a significant amount of time reading and understanding API documentations. On the other hand, system automation requires API calls to enable components to interact with each other. However, the complex structure of API documentations makes it hard to understand the wide variety of API documentations. By providing a machine-learning platform that can extract and standardize the API usage information creation of API-enabled systems, we can help developers easily use APIs in their applications. Although we targeted a specific group of documents, i.e. REST API documentations the novel methods and approaches we presented for designing an ML-based web crawler, content-based learning, signature-based learning, and deep content mapping, can all be applied to any natural language model that aims to extract knowledge from text data.

Bibliography

A. Abbasi, S. Sarker, and R. Chiang. Big data research in information systems: Toward an inclusive research agenda. *Journal of the Association for Information Systems*, 17(2):3, 2016.

D. Abbott. *Applied predictive analytics: Principles and techniques for the professional data analyst*. John Wiley & Sons, 2014.

S. Aerts, D. Lambrechts, S. Maity, P. Van Loo, B. Coessens, F. De Smet, L.-C. Tranchevent, B. De Moor, P. Marynen, B. Hassan, et al. Gene prioritization through genomic data fusion. *Nature biotechnology*, 24(5):537, 2006.

D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 530–533. ACM, 2011.

A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC bioinformatics*, 9(11):S2, 2008.

A. Alexandridis, P. Patrinos, H. Sarimveis, and G. Tsekouras. A two-stage evolutionary algorithm for variable selection in the development of rbf neural network models. *Chemometrics and intelligent laboratory systems*, 75(2):149–162, 2005.

M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*, 2017.

S. A. Ambrose, M. W. Bridges, M. DiPietro, M. C. Lovett, and M. K. Norman. *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons, 2010.

A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the web. *ACM Transactions on Internet Technology (TOIT)*, 1(1):2–43, 2001.

J. Archenaa and E. M. Anita. Interactive big data management in healthcare using spark. In *Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16')*, pages 265–272. Springer, 2016.

M. Armbrust, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin, and M. Zaharia. Scaling spark in the real world: performance and usability. *Proceedings of the VLDB Endowment*, 8(12):1840–1843, 2015.

S. Arora. Analyzing mobile phone usage using clustering in spark mllib and pig. *International Journal*, 8(1), 2017.

M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti. Big data machine learning using

apache spark mllib. In *2017 ieee international conference on big data (big data)*, pages 3492–3498. IEEE, 2017.

S. Bahassine, A. Madani, M. Al-Sarem, and M. Kissi. Feature selection using an improved chi-square for arabic text classification. *Journal of King Saud University-Computer and Information Sciences*, 2018.

M. Bahrami and W.-P. Chen. Automated web service specification generation through a transformation-based learning. In *International Conference on Services Computing*. Springer, 2020.

M. Bahrami, A. Khan, and M. Singhal. An energy efficient data privacy scheme for iot devices in mobile cloud computing. In *2016 IEEE International Conference on Mobile Services (MS)*, pages 190–195. IEEE, 2016.

M. Bahrami, J. Park, L. Liu, and W.-P. Chen. Api learning: Applying machine learning to manage the rise of api economy. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 151–154. International World Wide Web Conferences Steering Committee, 2018.

R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4):537–550, 1994.

Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Y. Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

K. K. Bharti and P. K. Singh. Hybrid dimension reduction by integrating feature selection with feature extraction method for text clustering. *Expert Systems with Applications*, 42(6):3105–3114, 2015.

H. S. Bhat, R. Madushani, and S. Rawat. Scalable sde filtering and inference with apache spark,". *Journal of Machine Learning Research W&CP*, 2016.

A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1-2):245–271, 1997.

R. R. Bouckaert, E. Frank, M. A. Hall, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. Weka—experiences with a java open-source project. *The Journal of Machine Learning Research*, 11:2533–2541, 2010.

C. Castillo. Effective web crawling. In *Acm sigir forum*, volume 39, pages 55–56. Acm, 2005.

G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

W. W. W. Consortium et al. Html 4.01 specification. 1999.

O. Cordón, S. Damas, and J. Santamaría. Feature-based image registration by means of the chc evolutionary algorithm. *Image and Vision Computing*, 24(5): 525–533, 2006.

M. Daniels, P. Ghadimi, I. Lanning, C. Heavey, A. Ryan, and M. Southern. An engineering prototype workflow management system. *IFAC Proceedings Volumes*, 46(9):1471–1476, 2013.

L. Delèger, R. Bossy, E. Chaix, M. Ba, A. Ferrė, P. Bessieres, and C. Nėdellec. Overview of the bacteria biotope task at bionlp shared task 2016. In *Proceedings of the 4th BioNLP Shared Task Workshop*, pages 12–22, 2016.

D. Ding, D. Wu, and F. Yu. An overview on cloud computing platform spark for human genome mining. In *Mechatronics and Automation (ICMA), 2016 IEEE International Conference on*, pages 2605–2610. IEEE, 2016.

Z. Ding, R. Xia, J. Yu, X. Li, and J. Yang. Densely connected bidirectional lstm with applications to sentence classification. *arXiv preprint arXiv:1802.00889*, 2018.

L. J. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of genetic algorithms*, volume 1, pages 265–283. Elsevier, 1991.

R. Fang, S. Pouyanfar, Y. Yang, S.-C. Chen, and S. Iyengar. Computational health informatics in the big data age: a survey. *ACM Computing Surveys (CSUR)*, 49(1):12, 2016.

M. S. Fazli, S. A. Vella, S. N. Moreno, and S. Quinn. Computational motility tracking of calcium dynamics in toxoplasma gondii. *arXiv preprint arXiv:1708.01871*, 2017.

R. T. Fielding. Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000.

R. T. Fielding, R. N. Taylor, J. R. Erenkrantz, M. M. Gorlick, J. Whitehead, R. Khare, and P. Oreizy. Reflections on the rest architectural style and" principled design of the modern web architecture"(impact paper award). In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 4–14, 2017.

J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

G. Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of machine learning research*, 3(Mar):1289–1305, 2003.

M. Frampton. *Mastering Apache Spark*. Packt Publishing Ltd, 2015.

K. Fundel, R. Küffner, and R. Zimmer. Relex—relation extraction using dependency parse trees. *Bioinformatics*, 23(3):365–371, 2006.

A. Gandomi and M. Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144, 2015.

A. Garcıa-Pablos, M. Cuadros, and G. Rigau. V3: Unsupervised aspect based sentiment analysis for semeval-2015 task 12. *SemEval-2015*, page 714, 2015.

D. E. Goldberg. Genetic alogorithms in search. *Optimization & Machine Learning*, 1989.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

S. Gopalani and R. Arora. Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications*, 113(1), 2015.

Z. GuoDong, S. Jian, Z. Jie, and Z. Min. Exploring various knowledge in relation extraction. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 427–434. Association for Computational Linguistics, 2005.

H. Gurulingappa, A. M. Rajput, A. Roberts, J. Fluck, M. Hofmann-Apitius, and L. Toldo. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of biomedical informatics*, 45(5):885–892, 2012.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

M. Habibi, L. Weber, M. Neves, D. L. Wiegandt, and U. Leser. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics*, 33(14):i37–i48, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A. Holzinger and I. Jurisica. Knowledge discovery and data mining in biomedical informatics: The future is in integrative, interactive machine learning solutions. In *Interactive knowledge discovery and data mining in biomedical informatics*, pages 1–18. Springer, 2014.

I. a. Q. M. J. How, D. I. M. H. I. Leave, Y. S. I. F. Question, and T. Y. Why. A case study: Apache spark.

A. Hryhorzhevska, M. Wiewiórka, M. Okoniewski, and T. Gambin. Scalable framework for the analysis of population structure using the next generation sequencing data. In *International Symposium on Methodologies for Intelligent Systems*, pages 471–480. Springer, 2017.

A. Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

H. Ji, S. H. Weinberg, M. Li, J. Wang, and Y. Li. An apache spark implementation of block power method for computing dominant eigenvalues and eigenvectors of large-scale matrices. In *Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on*, pages 554–559. IEEE, 2016.

T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning Proceedings 1994*, pages 121–129. Elsevier, 1994.

N. Kang, B. Singh, C. Bui, Z. Afzal, E. M. van Mulligen, and J. A. Kors. Knowledge-based extraction of adverse drug events from biomedical text. *BMC bioinformatics*, 15(1):64, 2014.

M. F. Karaca and S. Bayir. Examining the impact of feature selection methods on text classification. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 8(12):380–388, 2017.

R. Kennedy. J. and eberhart, particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks IV, pages*, volume 1000, 1995.

Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

J. Kivinen, M. K. Warmuth, and P. Auer. The perceptron algorithm versus winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1-2):325–343, 1997.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

D. Koller and M. Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

N. Kwak and C.-H. Choi. Input feature selection for classification problems. *IEEE transactions on neural networks*, 13(1):143–159, 2002.

P. Langley et al. Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall symposium on relevance*, volume 184, pages 245–271, 1994.

L. Laura and G. Me. Searching the web for illegal content: the anatomy of a semantic search engine. *Soft computing*, 21(5):1245–1252, 2017.

M. H. Law, M. A. Figueiredo, and A. K. Jain. Simultaneous feature selection and clustering using mixture models. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1154–1166, 2004.

C. Lazar, J. Taminau, S. Meganck, D. Steenhoff, A. Coletta, C. Molter, V. de Schaetzen, R. Duque, H. Bersini, and A. Nowe. A survey on filter techniques for feature selection in gene expression microarray analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 9(4): 1106–1119, 2012.

R. Leaman and G. Gonzalez. Banner: an executable survey of advances in biomedical named entity recognition. In *Biocomputing 2008*, pages 652–663. World Scientific, 2008.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

M.-S. Lee, E. Kim, C.-S. Nam, and D.-R. Shin. Design of educational big data application using spark. In *Advanced Communication Technology (ICACT), 2017 19th International Conference on*, pages 355–357. IEEE, 2017.

U. Leser and J. Hakenberg. What makes a gene name? named entity recognition in the biomedical literature. *Briefings in bioinformatics*, 6(4):357–369, 2005.

F. Li, M. Zhang, G. Fu, and D. Ji. A neural joint model for entity and relation extraction from biomedical text. *BMC bioinformatics*, 18(1):198, 2017.

G. Li, K. E. Ross, C. N. Arighi, Y. Peng, C. H. Wu, and K. Vijay-Shanker. mirtex: a text mining system for mirna-gene relation extraction. *PLoS computational biology*, 11(9):e1004391, 2015.

L. Li, L. Jin, Y. Jiang, and D. Huang. Recognizing biomedical named entities based on the sentence vector/twin word embeddings conditioned bidirectional lstm. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, pages 165–176. Springer, 2016.

Q. Li and H. Ji. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 402–412, 2014.

F. Liang, C. Feng, X. Lu, and Z. Xu. Performance benefits of datampi: a case study with bigdatabench. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 111–123. Springer, 2014.

N. Limsopatham and N. Collier. Learning orthographic features in bi-directional lstm for biomedical named entity recognition. In *Proceedings of the Fifth Workshop on Building and Evaluating Resources for Biomedical Text Mining (BioTxtM2016)*, pages 10–19, 2016a.

N. Limsopatham and N. H. Collier. Bidirectional lstm for named entity recognition in twitter messages. 2016b.

C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin. Large-scale logistic regression and linear support vector machines using spark. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 519–528. IEEE, 2014.

Z. Lv, J. Chirivella, and P. Gagliardo. Bigdata oriented multimedia mobile health applications. *Journal of medical systems*, 40(5):120, 2016.

Z. Lv, H. Song, P. Basanta-Val, A. Steed, and M. Jo. Next-generation big data analytics: State of the art, challenges, and future research topics. *IEEE Transactions on Industrial Informatics*, 13(4):1891–1899, 2017.

X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

R. Mackin. On collocations: Words shall be known by the company they keep. *Honour of as Hornby*, pages 149–165, 1978.

M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller, and S. Ceri. Genometric query language: a novel approach to large-scale genomic data management. *Bioinformatics*, 31(12):1881–1888, 2015.

P. Meesad, P. Boonrawd, and V. Nuipian. A chi-square-test for word importance differentiation in text classification. In *Proceedings of International Conference on Information and Electronics Engineering*, pages 110–114, 2011.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016a.

X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al. Mllib: Machine learning in apache spark. *JMLR*, 17(34):1–7, 2016b.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013c.

M. Miwa and M. Bansal. End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*, 2016.

D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 234–241. ACM, 2004.

D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

S. S. Nambisan, J. E. Alleman, S. Q. Larson, and M. G. Grogg. Pilot initiative in iowa for an intern development and management program. *Transportation research record*, 2414(1):35–44, 2014.

J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

N. T. Nguyen and Y. Tsuruoka. Extracting bacteria biotopes with semi-supervised named entity recognition and coreference resolution. In *Proceedings of the BioNLP Shared Task 2011 Workshop*, pages 94–101. Association for Computational Linguistics, 2011.

M. A. Nielsen. *Neural networks and deep learning*, volume 25. Determination press USA, 2015.

M. Parashar, X. Li, and S. Chandra. *Advanced computational infrastructures for parallel and distributed applications*, volume 66. John Wiley & Sons, 2010.

R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

M. Pecaric, K. Boutis, J. Beckstead, and M. Pusic. A big data and learning analytics approach to process-level feedback in cognitive simulations. *Academic Medicine*, 92(2):175–184, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011a.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011b.

R. Pita, C. Pinto, P. Melo, M. Silva, M. Barreto, and D. Rasella. A spark-based workflow for probabilistic record linkage of healthcare data. In *EDBT/ICDT Workshops*, pages 17–26, 2015.

S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, and T. Salakoski. Bioinfer: a corpus for information extraction in the biomedical domain. *BMC bioinformatics*, 8(1):50, 2007.

W. Raghupathi and V. Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):3, 2014.

A. Rios and R. Kavuluru. Convolutional neural networks for biomedical text classification: application in indexing biomedical articles. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 258–267. ACM, 2015.

C. Rong et al. Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 565–569. IEEE, 2011.

D. Roth and W.-t. Yih. Global inference for entity and relation identification via a linear programming formulation. *Introduction to statistical relational learning*, pages 553–580, 2007.

J. Ryan. Rapidminer for text analytic fundamentals. *Text Mining and Visualization: Case Studies Using Open-Source Tools*, 40:1, 2016.

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

G. Salton and C.-S. Yang. On the specification of term values in automatic indexing. *Journal of documentation*, 29(4):351–372, 1973.

N. Sánchez-Maroño, A. Alonso-Betanzos, and M. Tombilla-Sanromán. Filter methods for feature selection–a comparative study. In *International Confer-*

ence on Intelligent Data Engineering and Automated Learning, pages 178–187. Springer, 2007.

Y. Sasaki, D. Kawai, and S. Kitamura. The anatomy of tweet overload: How number of tweets received, number of friends, and egocentric network density affect perceived information overload. *Telematics and Informatics*, 32(4):853–861, 2015.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.

I. Segura-Bedmar, P. Martínez, and M. H. Zazo. Semeval-2013 task 9: Extraction of drug-drug interactions from biomedical texts (ddiextraction 2013). In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, volume 2, pages 341–350, 2013.

S. Sharma and P. Gupta. The anatomy of web crawlers. In *International Conference on Computing, Communication & Automation*, pages 849–853. IEEE, 2015.

A. G. Shoro and T. R. Soomro. Big data analysis: Apache spark perspective. *Global Journal of Computer Science and Technology*, 15(1), 2015.

R. Shyam, B. G. HB, S. Kumar, P. Poornachandran, and K. Soman. Apache spark a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska. Mli: An api for distributed machine learning. In *2013 IEEE 13th International Conference on Data Mining*, pages 1187–1192. IEEE, 2013.

E. R. Sparks, A. Talwalkar, M. J. Franklin, M. I. Jordan, and T. Kraska. Tupaq: An efficient planner for large-scale predictive analytic queries. *arXiv preprint arXiv:1502.00068*, 2015a.

E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380. ACM, 2015b.

E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380. ACM, 2015c.

M. Syed. *Using apache spark for scalable gene sequence analysis*. PhD thesis, TEXAS A&M UNIVERSITY-COMMERCE, 2016.

A. P. Tafti, J. Badger, E. LaRose, E. Shirzadi, A. Mahanke, J. Mayer, Z. Ye, D. Page, and P. Peissig. Adverse drug event discovery using biomedical literature: a big data neural network adventure. *JMIR Medical Informatics*, 2017a.

A. P. Tafti, E. Behravesh, M. Assefi, E. LaRose, J. Badger, J. Mayer, A. Doan, D. Page, and P. Peissig. bignn: an open-source big data toolkit focused on biomedical sentence classification. In *Proceedings of the IEEE BIG DATA*, 2017b.

A. P. Tafti, E. Behravesh, M. Assefi, E. LaRose, J. Badger, J. Mayer, A. Doan, D. Page, and P. Peissig. bignn: an open-source big data toolkit focused on biomedical sentence classification. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 3888–3896. IEEE, 2017c.

A. P. Tafti, E. LaRose, J. C. Badger, R. Kleiman, and P. Peissig. Machine learning-as-a-service and its application to medical informatics. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 206–219. Springer, 2017d.

L. Talavera. An evaluation of filter and wrapper methods for feature selection in categorical clustering. In *International Symposium on Intelligent Data Analysis*, pages 440–451. Springer, 2005.

D. Talia. Toward cloud-based big-data analytics. *IEEE Computer Science*, pages 98–101, 2013.

A. Tizghadam and A. Leon-Garcia. Application platform for smart transportation. In *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*, pages 26–32. Springer, 2015.

M. A. Uddin, J. bibi Joolee, A. Alam, and Y.-K. Lee. Human action recognition using adaptive local motion descriptor in spark. *IEEE Access*, 2017.

J. D. Van Horn. Opinion: Big data biomedicine offers big higher education opportunities. *Proceedings of the National Academy of Sciences*, 113(23):6322–6324, 2016.

V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

Z.-Y. Wang and H.-Y. Zhang. Rational drug repositioning by medical genetics. *Nature biotechnology*, 31(12):1080, 2013.

S. Wardley. Oscon 2014 keynote: "introduction to value chain mapping". URL https://www.youtube.com/watch?v=NnFeIt-uaEc.

C.-H. Wei, Y. Peng, R. Leaman, A. P. Davis, C. J. Mattingly, J. Li, T. C. Wiegers, and Z. Lu. Assessing the state of the art in biomedical relation extraction: overview of the biocreative v chemical-disease relation (cdr) task. *Database*, 2016, 2016.

M. S. Wiewiórka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M. J. Okoniewski. Sparkseq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics*, page btu343, 2014.

J. Yang, E. Wittern, A. T. Ying, J. Dolby, and L. Tan. Automatically ex-

tracting web api specifications from html documentation. *arXiv preprint arXiv:1801.08928*, 2018.

Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.

Z. Ye, A. P. Tafti, K. Y. He, K. Wang, and M. M. He. Sparktext: Biomedical text mining on big data framework. *PloS one*, 11(9):e0162721, 2016.

N. Yousefi, M. Georgiopoulos, and G. C. Anagnostopoulos. Multi-task learning with group-specific feature space sharing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 120–136. Springer, 2015.

L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of machine learning research*, 5(Oct):1205–1224, 2004.

M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: a unified

engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

X. Zhang and Y. LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

Z. Zheng, X. Wu, and R. Srihari. Feature selection for text categorization on imbalanced data. *ACM Sigkdd Explorations Newsletter*, 6(1):80–89, 2004.

X. Zhou, J. Menche, A.-L. Barabási, and A. Sharma. Human symptoms–disease network. *Nature communications*, 5:4212, 2014.