Enhancement of Natural Language to SQL Query Conversion using Machine Learning Techniques

Akshar Prasad[1], Sourabh S Badhya[2], Yashwanth YS[3], Shetty Rohan[4], Shobha G[5], Deepamala N[6]
Department of Computer Science and Engineering
RV College of Engineering
Bengaluru, India

Abstract—**In the age of information explosion, there is a huge data that is stored in the form of database and accessed using various querying languages. The major challenges faced by a user accessing this data is to learn the querying language and understand the various syntax associated with it. Query given in the form of Natural Language helps any naïve user to access database without learning the query languages. The current process of conversion of Natural Language to SQL Query using a rule-based algorithm is riddled with challenges -- identification of partial or implied data values and identification of descriptive values being the predominant ones. This paper discusses the use of a synchronous combination of a hybrid Machine Learning model, Elastic Search and WordNet to overcome the above-mentioned challenges. An embedding layer followed by a Long Short-Term Memory model is used to identify partial or implied data values, while Elastic Search has been used to identify descriptive data values (values which have lengthy data values and may contain descriptions). This architecture enables conversion systems to achieve robustness and high accuracies, by extracting meta data from the natural language query. The system gives an accuracy of 91.7% when tested on the IMDb database and 94.0% accuracy when tested on Company Sales database.**

Keywords—Machine learning; natural language to SQL query; long short-term memory; embedding layer; elastic search; hybrid architecture

I. Introduction

Availability of data and its analytics have revolutionized our life in all aspects. One of the popular methods of storing and accessing data is using Structured Query Language (SQL). SQL is a domain-specific programming language designed to store and access data in relational databases. It requires professional skills to use it. With the demand for data increasing exponentially, a simpler querying method which requires lesser or no learning time is a necessity. Hence, significant efforts are on to make Natural Language an interface between humans and the data stored in computers. Querying database using Natural Language makes data access simpler and affordable by all users.

Currently majority of the conversion systems that convert Natural Language to SQL query employ rule-based algorithms [1], [2]. One of the main challenges of this method is to identify implied or partial data values in the Natural Language. Another frequent failure case of such algorithms is the inability to capture the lengthy data values, often the attributes which contain descriptions and hence referred to as 'descriptive values' in this paper. The proposed system aims to resolve these shortcomings and increase the accuracy and robustness of the Natural Language to SQL query conversion systems.

To understand the challenge of identifying implied/partial data values in a domain specific database, consider the following example Natural Language query given to a sales database: 'Get the price of *product* red scooter'. This can be understood by a rule-based algorithm that 'red scooter' is a product and has to be searched in the corresponding 'product' attribute. However, the query 'Get the price of red scooter' requires the system to understand that scooter is a data value of 'product' attribute. Another interesting query is, 'Get the price of red two-wheeler' which requires the system to be intelligent and robust to understand that the user is implying the data-value 'scooter' with the use of the term 'two-wheeler'.

The proposed system uses an embedding layer followed by an LSTM model to pick up n-grams which are similar to determine a data value with a confidence greater than a pre-set threshold. The system has also been equipped with WordNet to find hyponyms of attributes to pick up implied values, in cases where the schema vocabulary is not expressive enough to train the embedding layer well.

Descriptive columns have been dealt with separately in this architecture. For example, let a database maintained by a pet adoption centre, have an entry with 'Animal Name' as 'Baxter' with 'Description' as 'He is a highly active and enthusiastic six-month old dog. He is black in colour and loves to chase vehicles'. A Natural Language query, 'Name the six-month dog which is fun loving and dark in colour' would be highly challenging to convert to the right SQL query, without special care being taken to understand the semantics. The proposed system uses Elastic Search to identify such 'descriptive values.

In all the existing systems, either an unintelligent rule-based system is adopted or the machine learning models are burdened with the entire task of conversion. The proposed system, which adapts a few concepts implemented in [10] and [11], although essentially a rule-based algorithm, uses machine learning models, WordNet and Elastic Search to enhance the conversion by overcoming the challenges faced by rule-based systems. Hence the system is more robust and the accuracy of conversion increases by an approximated 11-16%.

The paper describes the architecture and the techniques used by the proposed system and tests the performance of the system on the IMDb database [3]. To test the performance on 'descriptive data values', the system is tested on Company Sales database. It is to be noted that when tested on the same database used to test the SQLizer [4], the architecture's accuracy is 13.8% higher, considering SQLizer's 'Top 1' results.

II. Related Work

Conversion of Natural Language to SQL query was first explored in 1992, by Nikolaus Ott et al [5] where Natural Language inputs were mapped to an augmented SQL language. Different approaches to such conversions, some of which involve machine learning are discussed further.

S. Javubar *et al* has used standard natural language techniques such as morphological analysis, semantic analysis, mapping tables for retrieval of reports from social web data [1]. Xiaojun Xu *et al* in their paper 'SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning' attempt the conversion by filling in the slots of a standard SQL template with the data values present in the sentence using a CNN model. The performance is being considered on two different parameters, Query-Match with accuracy of 65.5% and Execution Accuracy of 71.5% [6].

Victor Zhong *et.al* in their paper 'Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning' use separate rules i.e. Neural Network models for each SQL clause. The different clauses considered are as follows

1) Aggregation clause – The tokens in the sentence are first mapped to its scalar attention score and these are normalized to obtain a distribution of input encodings. Sum of all the input encodings is taken and a multi-layer perceptron is used to convert the sum to a score corresponding to the aggregation (α). Softmax function is used to normalize the scores.

2) Select clause – Encoding of the column name with a LSTM and a multi-layer perceptron is applied over the column representations, conditioned on the input representation, to compute a score for each column.

3) Where clause – Reinforcement learning is applied to learn a policy to optimize execution results of expected correctness.

Finally, a mixed objective function is applied to combine the result of all the three clauses. The performance is considered on two different parameters, Query-Match Accuracy is 48.3% and Execution Accuracy of 59.4% [7].

Geordani et al. use concept of structured kernels e.g. Sequence and Tree Kernels which is referred to as structures that are created to classify the words present in the input query into appropriate tags. In this paper, a detailed comparison on different kernels were considered and appropriate combination were made to generate better results. Accuracies were considered for both datasets - Geo dataset (75.9%) and RestQueries dataset (84.7%) [8]. P. Utama *et al* in their paper 'An End-to-end Neural Natural Language Interface for Databases' use a novel concept of Neural Query Translation. An automatically generated dataset is fed into a Recurrent Neural Network and is used at runtime to convert natural language into SQL query. An Interactive auto-completion system increases translation accuracy since users enter less ambiguous sentences as input. Accuracies were considered for both datasets - Geo dataset – (48.6%) and Patients dataset (75.93%) [9].

F. Li *et al.* in their paper 'Constructing an interactive natural language interface for relational databases' explore the construction of parse trees and query trees to individually map words into their corresponding SQL tags. The system is commonly referred to as NaLIR (Natural Language Interface for Relational databases). An accuracy of 57.14% was achieved for the MAS dataset [2].

III. Proposed System

The proposed system extracts the dataset from the input relational database in CSV format. It must be noted that the system is not specific to a particular schema, but the models need to be re-trained if a new database is to be introduced. Three separate components work synchronously to extract maximum latent information from the dataset, which can either be used to enrich the natural language or be stored to use during conversion.

The system architecture has been described in Fig. 1. Each of the components described here is explained in detail in the following sections.

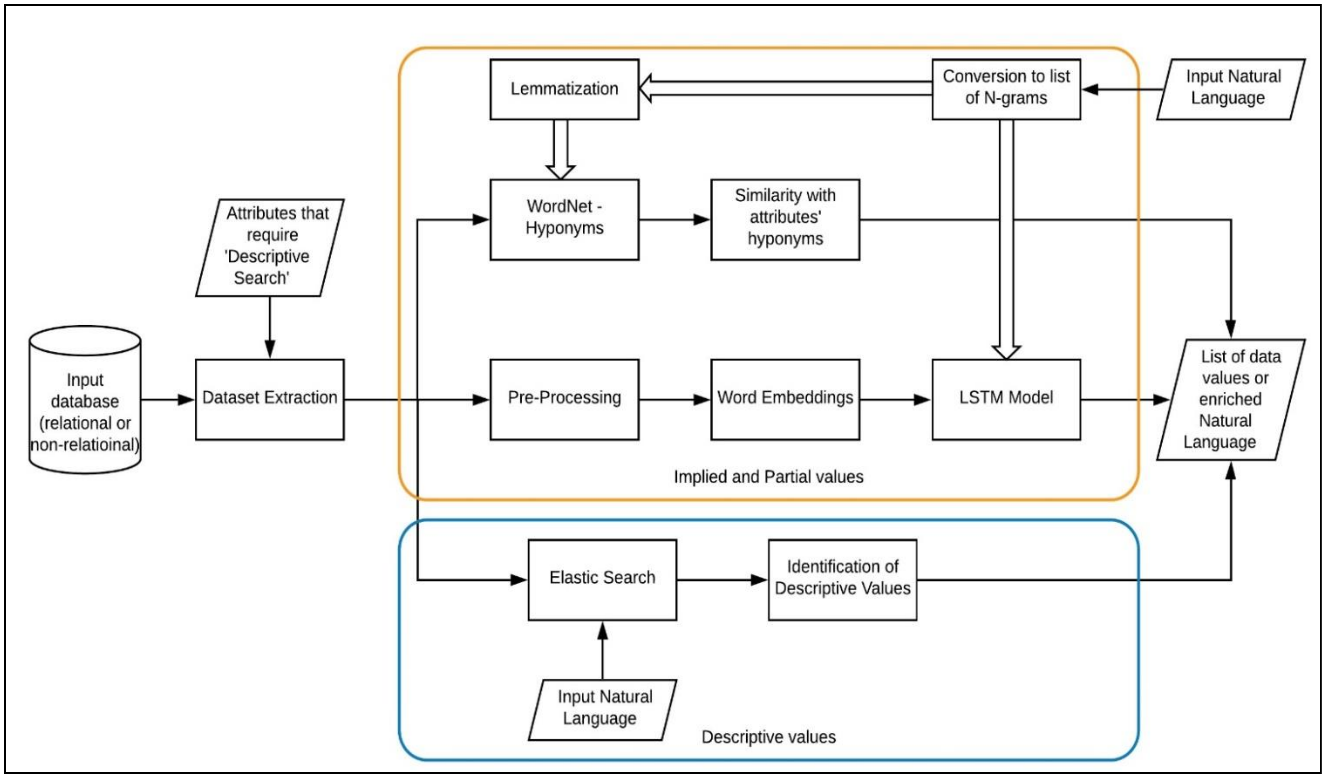

Input Natural Language
Conversion to list of N-grams
Lemmatization
WordNet - Hyponyms
Similarity with attributes' hyponyms
Attributes that require 'Descriptive Search'
Input database (relational or non-relatioinal)
Dataset Extraction
Pre-Processing
Word Embeddings
LSTM Model
List of data values or enriched Natural Language
Implied and Partial values
Elastic Search
Identification of Descriptive Values
Input Natural Language
Descriptive values

Fig. 1. System Architecture.

IV. Dataset Extraction

A major challenge faced while integrating machine learning into the architecture is to find a suitable dataset. The dataset is extracted from the input relational database. Apache Common CSV Library [12] has been used to extract the dataset in the form of CSV files. Along with the relational database, markers for attributes which contain descriptive values' (Ex: Experience, Description, etc.) is to be provided as input. Note that the system architecture is not schema specific and the ML model must be re-trained for a new database.

V. Partial and Implied Values

This section describes how the system identifies and extracts the partial and implied data values. This meta data can be used to either enrich the natural language input or be used directly by the rule-based conversion algorithm.

A. Pre-Processing Techniques

1) Reconstruction of dataset: The dataset extracted in the earlier step is a flat file containing the data values for all the attributes. A temporary feature vector is formed by obtaining the unique values of each attribute.

The data value and its attribute name separated by a placeholder value is used as the target value for each sample. This ensures that an extensive search to determine the attribute of the identified partial/implied value is avoided later.

2) Splitting, Tokenization and One-hot encoding: It was observed that the patterns in data values are best captured when the data value is split into words and each word is split into n-grams (value of n is determined by length of word). Intuitively, the syllables of each word of a data value are used as the features.

Each unique split (syllable) in the dataset is designated an integer (tokenized) as the model accepts only integer values. (The samples now consist of variable number of integers.) A label encoder followed by a one hot encoder is used to convert the target vector to an integer binary matrix.

3) Random sampling: To make the model robust and to expose the word embedding layer to a more varied vocabulary, a random sampling technique for augmentation is employed.

Iterating through each sample, a variable number of tokens/integers (the range of which is calculated based on the length of the sample) is randomly chosen for a fixed number of times (based on the number of samples) and appended to the dataset along with its target value (which is a binary vector). It is to be noted that random sampling does not affect the ordering of the remaining tokens.

4) Padding and Truncation: To ensure that all samples are of uniform length, the samples are padded with trailing zeros. The samples are then truncated to the 75th quartile of lengths to ensure that most features are not the padded zeros. Repeating the features of data values which have very few tokens increased the accuracy of detection of such attributes/data values.

The pre-processing steps have been summarized in Fig. 2.

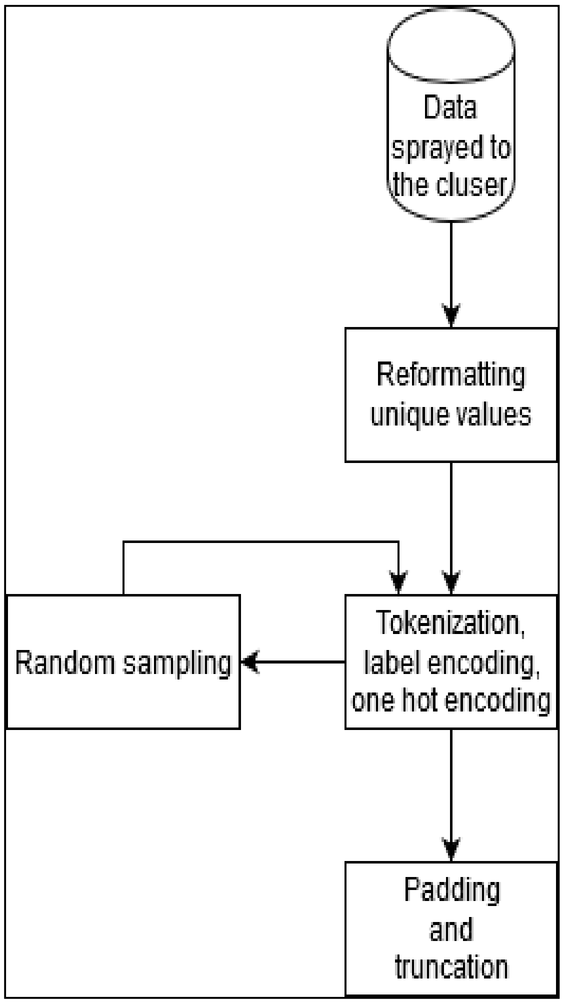

Data sprayed to the cluser
Reformatting unique values
Tokenization, label encoding, one hot encoding
Random sampling
Padding and truncation

Fig. 2. Pre-Processing Summary.

B. Embedding Layer

Word embeddings are representations of words in an n-dimensional vector space. This representation has helped bridge the gap between machine's and human brains' understanding of the language. An embedding layer maps each word in the given corpus to a dense vector which represent the projection of the word into a specified dimensional space. The vectors for the words are learnt based on its surrounding words in the given corpus. As a result, the vectors of words with similar meaning will be 'close to each other'. For example, the Euclidean distance between the vectors representing 'school' 'college' will be much lesser than between vectors representing 'school' and 'dog'.

Hence this approach succeeds in capturing the context of a word or sentence. This can be observed in Fig. 3, where the verb tense relationships and country-capital relationships have been recognized by the trained embedding layer.

C. Long Short-Term Memory (LSTM)

LSTM is a variant of Recurrent Neural Networks (RNN). RNNs which succeeded in creating a perception of storage or memory, often failed due to exploding and vanishing gradients. This failure case was overcome by LSTM with the use of a memory cell. The memory cell contains the current memory of the node which can be written into, read and erased just like a computer memory. (Note: This memory is analogous.)

The current timestep and the previous timestep's output is fed as an input to the LSTM node as seen in Fig. 4. The node contains a memory cell and four simple one-layer neural networks. While one neural network generates the new memory, two other neural networks control the significance given to the old memory and the new memory, and the other neural network generates the output from the new memory. Note that a 'tanh' activation function is used for memory generation, while a sigmoid activation is used to determine the significance.

Since LSTM specializes in processing series of samples where the temporal locality carries great significance, it has been used in this architecture to understand a series of dense vectors and classify them into the correct data value.

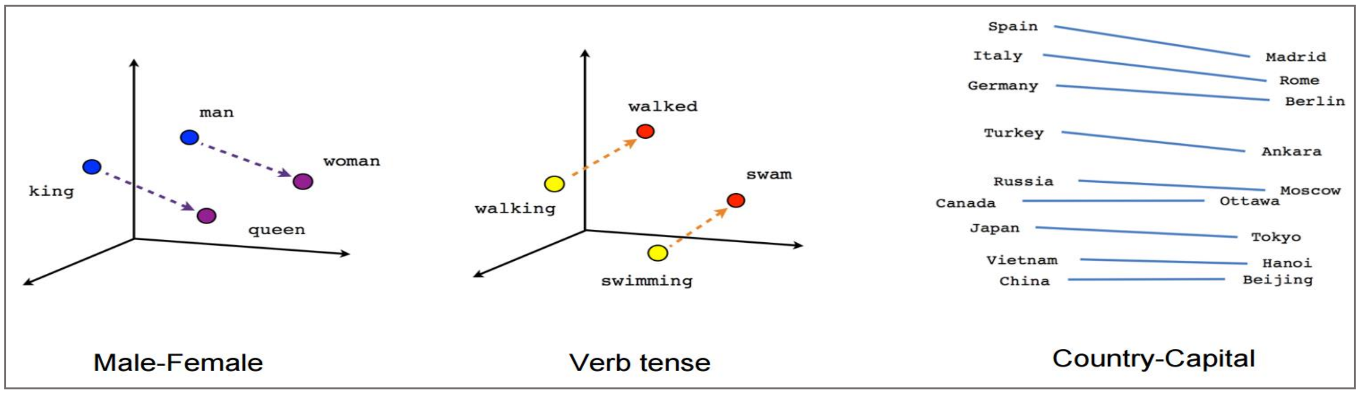


Fig. 3. Embedding Layer [13].

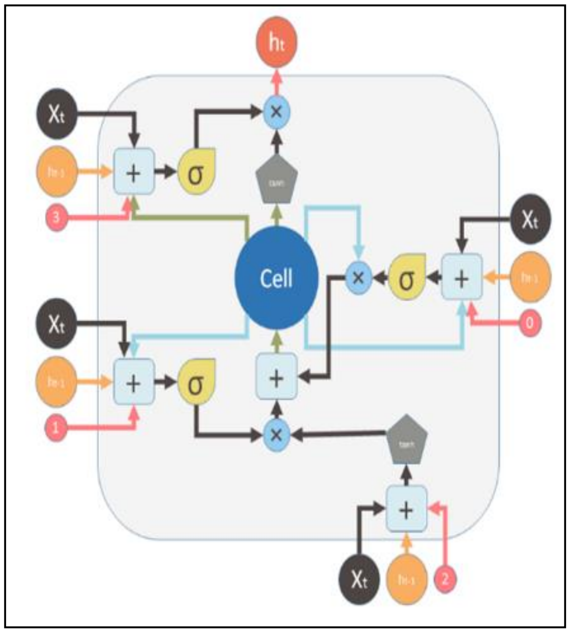


Fig. 4. Long Short-Term Memory [14].

The number of nodes used and the number of epochs are relatively small because of the following reasons:

1) Due to the nature of the dataset, there are high chances of over-fitting.

2) Speed of training the model must be high, as the model is re-trained every time a new schema is introduced.

D. Classification of Inputs

The input Natural Language query is tokenized and split into different sequences. Sequences of 1 word (1-gram) up to sequences of n words (n-gram, where n is determined by the number of tokens) is considered for prediction. The largest sequences and its classification are considered (i.e., sub-sequences are ignored).

The final, high confidence classifications given by the LSTM model can be used in multiple ways, couple of them are outlined below:

1) Enrich the natural language query: Replace the classified data value with the incomplete/implied data value (largest n-gram) and precede the attribute before the data value.

2) Store the data values and attribute names: Store the data value- attribute pair using a HashMap or any list and use it as required by the conversion algorithm.

VI. Descriptive Values

This section describes the elastic search approach taken for identifying and mapping the descriptive data values. These values are lengthy and tend to have high degrees of incompleteness and implications.

A modified version of the embedding layer and LSTM model approach was tried for such values as well. The approach's accuracy exceeds the elastic search approach only if the data values' vocabulary is expressive enough to develop a suitable vector space while training the embedding layer. As this is rare, the Elastic Search approach has been explored in this paper.

A. Architecture

Descriptive sentences are those sentences that provide broader context to the query. The extracted CSV file is used to create an index in Elastic Search. Elastic Search's Bulk API [15] provides the necessary functions that can create and store large data simultaneously. The 'helpers' module [16] in Python which is one of the ways to access this API is being used in this case to create and store the data from the .csv file. The benefit of using Bulk API is that it indexes chunks of data at once rather than one after the other. The input query is made to pass through the following to get the required information:

1) Analyzers: Analyzers are used to process the input query to a desired format. The analyzers are a class of in-built functions that bring the data to a standard format. Elastic Search has in-built analyzers such as Stop analyzer and Standard analyzer. Custom analyzers can be built to comply with the needs of the application. The system in this case makes use of 'Stop' and 'English Language' Analyzer [17], [18].

a) Stop Analyzer: This analyzer splits the text into individual words and removes all the pre-defined stop-words which are defined for English. Stop Analyzer also lower cases the remaining words after the removal of stop words. Elastic Search also supports Stop Analyzers in multiple languages.

Ex: *Input*: Get the doctors with master's degree.

Analyzer: Get doctors master's degree

It is to be noted that 'with' and 'the' tokens are discarded.

b) English Language Analyzer: This analyzer converts the words of the input query to its root word. This is used when all the words in the input query have different tenses. This analyzer brings them to their basic form. This makes the processing of the remaining words easier.

Ex: *Input*: Show all products which are red bikes.

Analyzer: Show all product which road bike.

It is to be noted that 'bikes' has changed to 'bike' and 'products' has changed to 'product'.

2) Searching through multiple attributes: Multiple columns can be searched in Elastic Search. This can be done by using the "multi_match" keyword [19]. The JSON request body which is to be sent to Elastic Search server can be written as follows:

```
{ "query":
        { "multi_match":
                { "query": input query,
                 "fields":[list of descriptive column names];
                }
        }
}
```

Format: Query for multiple attributes

The input query mentioned in the JSON request is the output obtained after the input query has been processed from the analyzers. “fields” represents the set of column names through which descriptive information needs to be searched for.

Ex: *Input*: Show all items which are road bikes.

Analyzer: show all product which road bike.

The request query is as follows:

```
{ “query”:
          { “multi_match”:
                      { “query”:“ list all item which road bike”,
                       “fields”: [‘UserDescription’,
‘ItemDescription’];
                      }
          }
}
```

Example: The input query is being searched across:

‘UserDescription’ and ‘ItemDescription’ fields of the index.

Once the request is sent, the response is received in the form of JSON. The JSON body contains useful information such as the “source” field which describes the appropriate description along with the field to which it's being matched.

In addition to this, there also exists a field called “_score” which indicates how relevant the description is to the matched description. The “_score” is calculated by Elastic Search by making use of a practical scoring function [20]. The practical scoring function is described in Elastics Search as follows:

$$score(q,d) : \\ = coord(q,d) \\ - queryNorm(q).?(tf(t\ in\ id).idf(t)^2.(t.getBoost()).norm(t,d))$$

Where, **t** refers to the term which is given in the input by the user, **q** is the cosine similarity between the vectors.

coord(q,d) represents a score dependent on how many query terms a given document contains.

tf(t in d) represents the term frequency of the term, t in the current document d.

queryNorm(q) normalizing factor to alter queries’ scores to make them comparable.

idf(t) stands for Inverse Document Frequency of t.

t.getBoost() is a search time boost of the term in the current query q.

norm(t,d) summarizes a few boost and length factors (indexing time).

Hence, this scoring function is dependent on the number of times a word in the input query appears in the documents that are present in the index. Intuitively, the relevance score of a descriptive value increases as the frequency of occurrence in documents increases.

Furthermore, a reasonable threshold for contextual relevance scoring must be set. Values above this limit are only to be considered for fieldname-value pair extraction.

3) Searching through multiple attributes: After the generation of pairs of fieldnames and values, a statement which corresponds to SQL can be written as follows-

WHERE $fieldname_1 = value_1$ AND $fieldname_2 = value_2$ AND.… $fieldname_N = value_N$;

Later, the following statement is appended to existing SQL output which does not handle descriptive or contextual columns. If there is no suitable pair then the statement is not generated and there is no effect on the output.

A summary of these steps is depicted in Fig. 5.

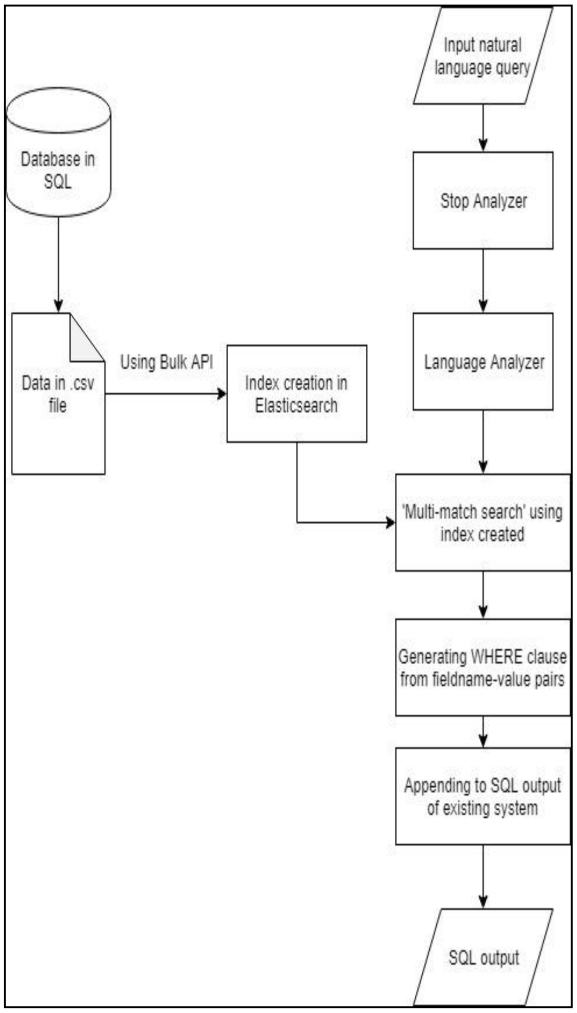
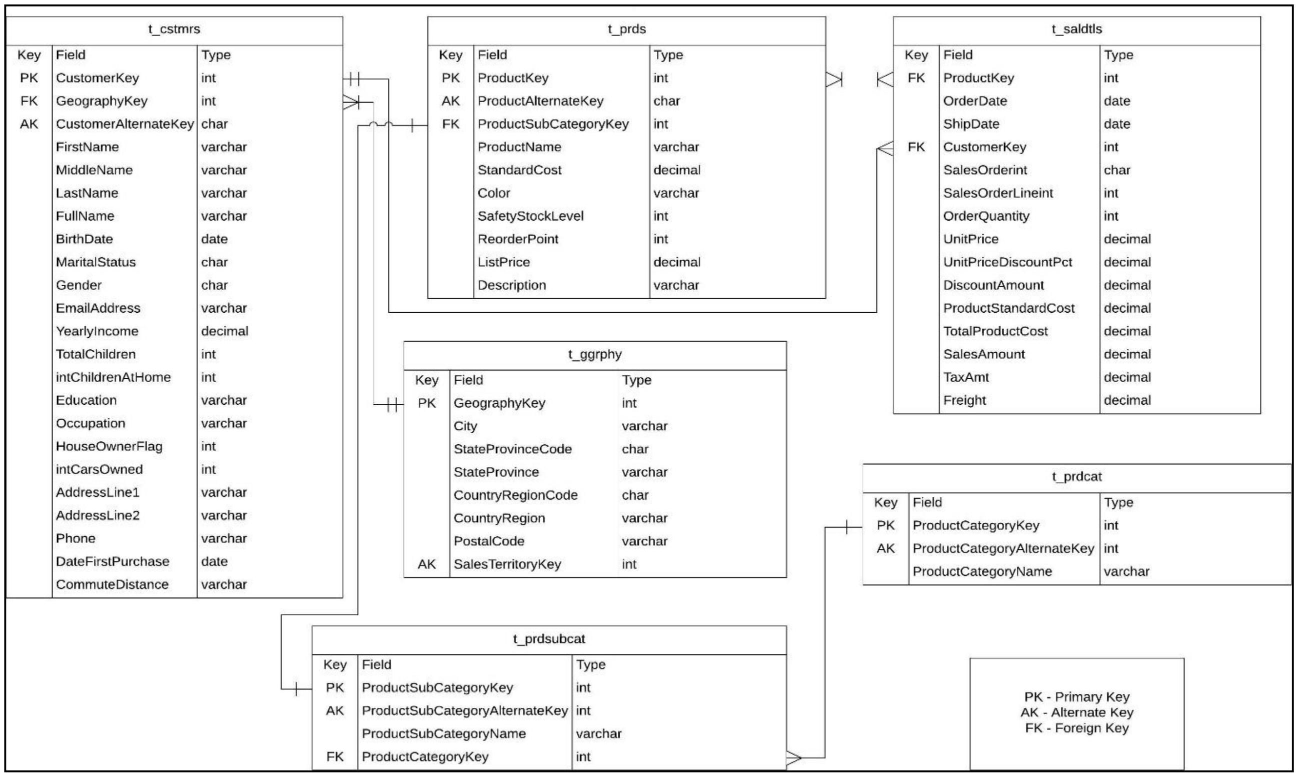

Input natural language query
Database in SQL
Stop Analyzer
Data in .csv file
Using Bulk API
Index creation in Elasticsearch
Language Analyzer
'Multi-match search' using index created
Generating WHERE clause from fieldname-value pairs
Appending to SQL output of existing system
SQL output

Fig. 5. System Architecture – Descriptive Values.

VII. RESULTS

A. Company Sales Database

The system is tested on the Company Sales database, whose schema is depicted in Fig. 6. The database has six tables which contain the data of products, customers and its sales.



t_cstmrs

Key	Field	Type
PK	CustomerKey	int
FK	GeographyKey	int
AK	CustomerAlternateKey	char
	FirstName	varchar
	MiddleName	varchar
	LastName	varchar
	FullName	varchar
	BirthDate	date
	MaritalStatus	char
	Gender	char
	EmailAddress	varchar
	YearlyIncome	decimal
	TotalChildren	int
	intChildrenAtHome	int
	Education	varchar
	Occupation	varchar
	HouseOwnerFlag	int
	intCarsOwned	int
	AddressLine1	varchar
	AddressLine2	varchar
	Phone	varchar
	DateFirstPurchase	date
	CommuteDistance	varchar

t_prds

Key	Field	Type
PK	ProductKey	int
AK	ProductAlternateKey	char
FK	ProductSubCategoryKey	int
	ProductName	varchar
	StandardCost	decimal
	Color	varchar
	SafetyStockLevel	int
	ReorderPoint	int
	ListPrice	decimal
	Description	varchar

t_saldtls

Key	Field	Type
FK	ProductKey	int
	OrderDate	date
	ShipDate	date
FK	CustomerKey	int
	SalesOrderint	char
	SalesOrderLineint	int
	OrderQuantity	int
	UnitPrice	decimal
	UnitPriceDiscountPct	decimal
	DiscountAmount	decimal
	ProductStandardCost	decimal
	TotalProductCost	decimal
	SalesAmount	decimal
	TaxAmt	decimal
	Freight	decimal

t_ggrphy

Key	Field	Type
PK	GeographyKey	int
	City	varchar
	StateProvinceCode	char
	StateProvince	varchar
	CountryRegionCode	char
	CountryRegion	varchar
	PostalCode	varchar
AK	SalesTerritoryKey	int

t_prdcat

Key	Field	Type
PK	ProductCategoryKey	int
AK	ProductCategoryAlternateKey	int
	ProductCategoryName	varchar

t_prdsubcat

Key	Field	Type
PK	ProductSubCategoryKey	int
AK	ProductSubCategoryAlternateKey	int
	ProductSubCategoryName	varchar
FK	ProductCategoryKey	int

Fig. 6. ER Diagram of Company Sales Database.

1) Experimental Setup: The tables ‘t_prds’ contains product data, ‘t_cstmrs’ contains customer data, ‘t_saldtls’ contains sales data, ‘t_prdcat’ contains product-category data, ‘t_prdsubcat’ contains product-subcategory data and ‘t_ggrphy’ contains the geographical location of the customers.

The attribute ‘Description’ in ‘t_prds’ table contain ‘descriptive values’ which describes each product. This column is marked while giving the database as input to the system.

2) Partial and Implied Values – Results: Of the 50 natural language queries that were designed to test this feature, 48 queries correctly identified the partial/implied data values and 47 queries were mapped to the correct SQL query.

The queries tested the ability of the system to generate queries with clauses such as WHERE, BETWEEN, HAVING, ORDER BY, GROUP BY, COUNT, SUM, MAX, MIN (aggregate functions).

Table I shows some of the correctly mapped natural language queries. The data values which were partial or implied have been highlighted in bold and the identified attributes have been underlined.

3) Descriptive Values – Results: The system was tested for 50 descriptive inputs and all the inputs mapped to appropriate descriptions and 49 of them formed to the correct SQL query. Results that were used for testing are shown in Table II. The descriptive data values have been highlighted in bold.

E. IMDB Database

The system is tested on the IMDb database whose schema is shown in Fig. 7. The database has tables that contain data of movies and its genres, actors, and directors.

1) Experimental Setup: The tables names are self-explanatory and the database overall contains movies’, actors’ and directors’ data and are linked through ‘roles’ and ‘movies_directors’ tables.

The genres of movies and directors are also contained in the database.

2) Partial and Implied Values – Results: Out of the 48 natural language queries that were designed to test this feature, 44 queries correctly identified the partial/implied data values and mapped to the correct SQL query.

The queries again tested the ability of the system to generate queries with clauses such as WHERE, BETWEEN, HAVING, ORDER BY, GROUP BY, COUNT, SUM, MAX, MIN (aggregate functions).

Table III contains some of the correctly mapped natural language queries. The data values which were partial or implied have been highlighted in bold and the identified attributes have been underlined.

Note, the IMDb database has no scope for testing descriptive values. Company Sales database has been used for this purpose.

TABLE I. COMPANY SALES DATABASE: PARTIAL / IMPLIED VALUES

Input Natural Language Query	**Enriched Natural Language Query**	**Output SQL Query**
show all unmarried customers who are men	show all ***single*** Gender '***male***' customers	SELECT * FROM t_cstmrs WHERE LOWER(MaritalStatus) = 'single' AND LOWER(Gender) = 'male'
Names of customers who have graduated and from germany or france	FullName Names of customers who have Education '***graduate degree***' and from CountryRegion 'germany' or CountryRegion 'france'	SELECT t_cstmrs.FullName FROM t_cstmrs INNER JOIN t_ggrphy ON t_ggrphy.GeographyKey = t_cstmrs.GeographyKey WHERE LOWER (t_ggrphy.CountryRegion) = 'germany' OR LOWER (t_ggrphy.CountryRegion) = 'france') AND (LOWER(t_cstmrs.Education) = 'graduate degree')
get the price of red or dark helmet	get the price of Color 'red' or Color '***black***' ProductSubCategoryName 'helmet'	SELECT ListPrice , Color FROM t_prdsubcat INNER JOIN t_prds ON t_prdsubcat.ProductSubCategoryKey = t_prds.ProductSubCategoryKey WHERE LOWER(Color) = 'red' OR LOWER(Color) = 'black'
how much does tire tube cost	how much does ProductName '***road tire tube***' cost	SELECT ListPrice , ProductName FROM t_prds WHERE LOWER(ProductName) = 'road tire tube'
get the orders from new south wales australia	get the orders from StateProvince 'new south wales' CountryRegion 'australia'	SELECT t_saldtls.OrderQuantity, t_ggrphy.CountryRegion, t_ t_cstmrs.FullName , t_ggrphy.StateProvince FROM t_ggrphy INNER JOIN t_cstmrs ON t_cstmrs.GeographyKey = t_ggrphy.GeographyKey INNER JOIN t_saldtls ON t_cstmrs.CustomerKey = t_saldtls.CustomerKey WHERE LOWER(t_cstmrs.StateProvince) = 'new south wales' AND LOWER(t_ggrphy.CountryRegion) = 'australia'
show subtotal of orders for helmet	show subtotal of orders for ProductSubCategoryName '***helmet***'	SELECT SUM(t_saldtls.SalesOrderint) FROM t_prds INNER JOIN t_saldtls ON t_prds.ProductKey = t_saldtls.ProductKey WHERE LOWER(t_prds.ProductName) = 'helmet'
Names of customers in germany who purchased bicycle but not tires	FullName Names of customers in CountryRegion 'germany' who purchased ProductCategoryName '***bikes***' but not ProductSubCategoryName '***road tire tubes***'	SELECT t_cstms.FullName FROM t_cstmrs INNER JOIN t_ggrphy ON t_ggrphy.GeographyKey = t_cstmrs.GeographyKey INNER JOIN t_saldtls ON t_saldtls.CustomerKey = t_cstmrs.CustomerKey INNER JOIN t_prds ON t_prds.ProductKey = t_cstmrs.ProductKey INNER JOIN t_prdsubcat ON t_prdsubcat.ProductSubCategoryKey = t_prds.ProductSubCategoryKey WHERE LOWER(t_prdsubcat.ProductSubCategoryName) != 'road tire tubes' AND LOWER(t_prdsubcat.ProductSubCategoryName) = 'bikes' AND LOWER (t_ggrphy.CountryRegion) = 'germany'

TABLE II. COMPANY SALES DATABASE: DESCRIPTIVE VALUES

Input Natural Languge Query	**Output SQL Query**
list price of black products which is responsive, maneuverable, low budget and can go off-road.	SELECT t_prds.ListPrice FROM t_prds WHERE t_prds.Description = ***'This low budget bike gives a high performance.. It is maneuverable and responsive. Highly suitable to go off-road'*** AND (LOWER(t_prds.Color) = 'black'
Name the Road Series all round bike item for road and racing.	SELECT t_prds.ProductName FROM t_prdcat INNER JOIN t_prdsubcat ON t_prdcat.ProductCategoryKey = t_prdsubcat.ProductCategoryKey INNER JOIN t_prds ON t_prdsubcat.ProductSubCategoryKey = t_prds.ProductSubCategoryKey WHERE t_prds.Description = ***'Has specifications similar to our Road series bikes, but the frame size has been made to suit a woman. All round bike, suitable for road and racing'*** AND LOWER(t_prdcat.ProductCategoryName) = 'racing' AND LOWER(t_prdsubcat.ProductSubCategoryName) = 'bike')
Get the customers who have bought a bike which is comfortable, provides durable frame and high power	SELECT t_cstmrs.FullName FROM t_cstmrs CustomerKey INNER JOIN t_prds ON t_prds.ProductKey = t_saldtls.ProductKey INNER JOIN t_saldtls ON t_saldtls.CustomerKey = t_csrmrs. WHERE t_prds.Description = '***Has a lot of the same features as our high end bikes such as a comfortable, durable frame, and high power***';
Select the items with plush custom saddle and space for panniers	SELECT t_prds.ProductName FROM t_prds WHERE t_prds.Description = ***'The all new plush custom saddle gives you the extra comfort while riding. Newly designed carrier gives you the extra space to add panniers and luggage bags. It provides high stability even when fully loaded.'***
Name the pink products which is aerodynamic bike designed for a woman.	SELECT t_prds.ProductNam e FROM t_prds WHERE t_prds.Description = ***'The sleek aerodynamic body designed for a woman allows you to either race,cross-train, or just socialize. Advanced seat technology provides comfort all day.'*** AND (LOWER(t_prds.Color) = 'pink'
Name the items which are developed by Adventure Works Cycles Professional race team	SELECT t_prds.ProductName FROM t_prds WHERE t_prds.Description = ***'Tt posseses a very light weight heat-treated steel frame, and highly precise steering contro006C. Developed with the Adventure Works Cycles professional race team, is driven only by champions'***

Get the price of items which is a higher end mountain bike with rear suspension	SELECT t_prds.ListPrice FROM t_prds WHERE t_prds.Description = ***'With enhanced performance achieved by using the ground breaking SQ Frame, highly strong rear suspension, this is a higher end model moutain bike'***
Show the price of bikes meant for racing enthusiasts with low budget	SELECT t_prds.ListPrice FROM t_prdcat INNER JOIN t_prdsubcat ON t_prdcat.ProductCategoryKey = t_prdsubcat.ProductCategoryKey INNER JOIN t_prds ON t_prdsubcat.ProductSubCategoryKey = t_prds.ProductSubCategoryKey WHERE t_prds.Description = ***'It is a low budget super strong bike that can ride on all terrains while keeping you in complete control. It is suited for all you racing enthusiasts'*** AND (t_prdcat.ProductCategoryName = 'bikes')
Select an item with mountain wheel for entry-level rider.	SELECT * FROM t_prds WHERE t_prds.Description = ***'Replacement mountain wheel for entry-level rider.'***
Name the items which have pioneering frame technology as the HQ steel frame.	SELECT t_prds.ProductName FROM t_prds WHERE t_prds.Description ***= 'The same pioneering frame technology is used to give you the highest value as the HQ steel frame.'***

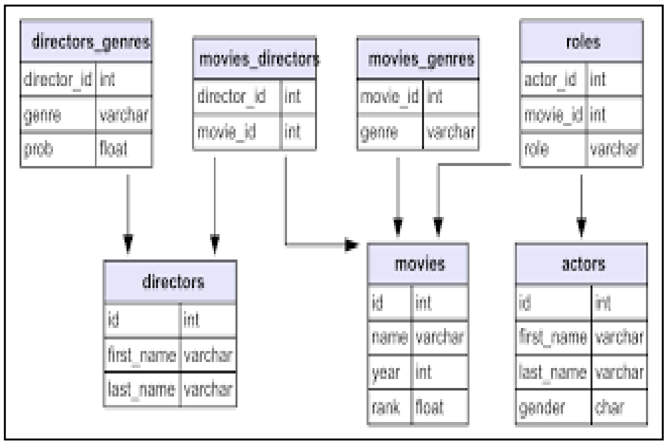


Fig. 7. IMDb Database Schema [21].

TABLE III. IMDB DATABASE: PARTIAL / IMPLIED VALUES

Input Natural Language Query	**Enriched Natural Language Query**	**Output SQL Query**
List the names of actors who played a lawyer	List the names of actors who played a <u>Roles.Role</u> '**Attorney'**	SELECT actors.first_name FROM actors INNER JOIN roles ON actors.id = roles.actor_id WHERE LOWER(roles.role) = 'attorney'
List the category of films enacted by Leonardo Dicaprio	List the **<u>movies.genre</u>** of films enacted by <u>actors.first_name</u> 'Leonardo' <u>actors.last_name</u> 'Dicaprio'	SELECT movies.genre FROM movies INNER JOIN roles ON movies.id = roles.movie_id INNER JOIN actors ON actors.id = roles.actor_id WHERE LOWER(actor.first_name) = 'leonardo' AND LOWER(actor.first_name) = 'dicaprio'
Name the satirical movies played by Frank Cady	Name the <u>movies.genre</u> **'Comedy'** movies played by <u>actors.first_name</u> 'Frank' <u>actors.last_name</u> 'Cady'	SELECT movies.name FROM movies INNER JOIN roles ON movies.id = roles.movie_id INNER JOIN actors ON actors.id = roles.actor_id WHERE LOWER(actor.first_name) = 'frank' AND LOWER(actors.last_name) = 'cady' AND LOWER(movies.genre) = 'comedy'
What are the fairy tale movies directed by Bill Condon	Name the <u>movies.genre</u> **'Fantasy'** movies played by <u>directors.first_name</u> 'Bill' <u>directors.last_name</u> 'Condon'	SELECT movies.name FROM movies INNER JOIN roles ON movies.id = roles.movie_id INNER JOIN directors ON directors.id = movies_directors.director_id WHERE LOWER(directors.first_name) = 'bill' AND LOWER(directors.last_name) = 'condon' LOWER(movies.genre) = 'fantasy')
who directed the movie name 10 Rillington Place	who directed the movie <u>movies.name</u> '10 Rillington Place'	SELECT directors.first_name , movies.name FROM movies INNER JOIN roles ON movies.id = roles.movie_id INNER JOIN actors ON actors.id = roles.actor_id INNER JOIN movies_directors ON movies.id = movies_directors.movie_id INNER JOIN directors ON directors.id = movies_directors.director_id where movie.name = '10 Rillington Place'
show the movies directed by Richard and enacted by Daniel	show the movies direced by <u>directors.first_name</u> 'Richard' and enacted by <u>actors.first_name</u> 'Daniel'	SELECT movies.name , directors.first_name FROM movies INNER JOIN roles ON movies.id = roles.movie_id INNER JOIN actors ON actors.id = roles.actor_id INNER JOIN movies_directors ON movies.id = movies_directors.movie_id INNER JOIN directors ON directors.id = movies_directors.director_id WHERE directors.first_name = 'Richard' AND actors.first_name = 'Daniel'

VIII. Conclusion

The proposed system addresses major challenges faced by many of the existing Natural Language to SQL Query conversion algorithms.

Partial and implied data values in the natural language queries are identified by a trained hybrid ML model. WordNet is also used as a safety net to understand implied data values where the vocabulary of the input relational database is not expressive. Descriptive values are identified with the help of Elastic Search.

Using the latent information gathered by the proposed architecture, the accuracy and the robustness of the Natural Language to SQL Query conversion system is proven to increase dramatically (11% to 16%). This has been demonstrated by extensively testing the system on the IMDb database as well as the Company Sales database.

The accuracy of the system is 91.7% on IMDb database and 94.0% on Company Sales database when tested on a diverse set of queries. This is a significantly higher performance compared to the discussed systems in section II.

This system can be used as a plug-in to any of the conversion systems being developed/used. The meta information extracted by the system helps developers boost the accuracy and robustness of their algorithm.

References

[1] Sathick KJ, Jaya A. Natural Language to SQL Generation for Semantic Knowledge Extraction in Social Web Sources. Indian Journal of Science and Technology. 2015; 8(1):1–10.

[2] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. PVLDB 2014, 8:73–84.

[3] IMDb data files available for download [Online]. Available: https://datasets.imdbws.com [Accessed: 19- Mar- 2020].

[4] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. Proc. ACM Program. Lang. 1, OOPSLA, Article 63 (October 2017), 26 pages. DOI:https://doi.org/10.1145/3133887.

[5] N. Ott, "Aspects of the automatic generation of SQL statements in a natural language query interface", Information Systems, vol. 17, no. 2, pp. 147-159, 1992.

[6] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. arXiv preprint ArXiv:1711.04436.

[7] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. arXiv Preprint arXiv:1709.00103 (2017).

[8] Giordani, A. and Moschitti, A. (2009b). Syntactic structural kernels for natural language interfaces to databases. In Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part I, ECML PKDD, Berlin, Heidelberg. Springer-Verlag.'09, pages 391–406.

[9] Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Cetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy and Arif Usta.An End-to-end Neural Natural Language Interface for Databases. arXiv Preprint arXiv:1804:00401.

[10] A. Prasad, G. Shobha, N. Deepamala, S. S. Badhya, Y. Yashwanth and S. Rohan, "Machine Learning Techniques to Understand Partial and Implied Data Values for Conversion of Natural Language to SQL Queries on HPCC Systems," 2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, 2019, pp. 1-5. doi: 10.1109/CSITSS47250.2019.9031035.

[11] S. S. Badhya, A. Prasad, S. Rohan, Y. S. Yashwanth, N. Deepamala and G. Shobha, "Natural Language to Structured Query Language using Elasticsearch for descriptive columns," 2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), Bengaluru, India, 2019, pp. 1-5. doi: 10.1109/CSITSS47250.2019.9031030.

[12] Apache Commons CSV API Documentation from the official website. https://commons.apache.org/proper/commons-csv/apidocs/index.html [Accessed: 19- Mar- 2020].

[13] R. Ruizendaal, "Deep Learning #4: Why You Need to Start Using Embedding Layers", Medium, 2019. [Online]. Available: https://towards datascience.com/deep-learning-4-embedding-layersf9a 02 d55ac12. [Accessed: 19- Mar- 2020].

[14] S. Yan, LSTM node. 2016. [Online]. Available: https://medium.com/ mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714. [Accessed: 19 - Mar- 2020].

[15] Bulk API, Elasticsearch Reference (7.2) in the official website. https://www.elastic.co/guide/en/elasticsearch/reference/current/docsbulk .html [Accessed: 19- Mar- 2020].

[16] Bulk Helpers, Python Elasticsearch Client in the official website. https://elasticsearch-py.readthedocs.io/en/master/helpers.html [Accessed: 19- Mar- 2020].

[17] Stop Analyzer, Elasticsearch Reference (7.2) in the official website. https://www.elastic.co/guide/en/elasticsearch/reference/current/analysi s-stop-analyzer.html [Accessed: 19- Mar- 2020].

[18] Language Analyzers, Elasticsearch Reference (7.2) in the official website. https://www.elastic.co/guide/en/elasticsearch/reference/current/ analysi s-lang-analyzer.html [Accessed: 19- Mar- 2020].

[19] Multi-match query, Elasticsearch Reference (7.2) in the official website. https://www.elastic.co/guide/en/elasticsearch/reference/current/querydsl-multi-match-query.html [Accessed: 19- Mar- 2020].

[20] TFIDFSimilarity (Lucene 4.6.0 API) in the official website. https://lucene.apache.org/core/4_6_0/core/org/apache/lucene/search/ similarities/TFIDFSimilarity.html [Accessed: 19- Mar- 2020].

[21] IMDb Dataset Schema [Online]. Available: http://kt.ijs.si/ janez_kranjc/ilp_datasets [Accessed: 19- Mar- 2020].