Codify: "Natural Language to Code, for Data Science Applications"

Abhishek Gupta
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
gupta.abhishek@fcrit.ac.in

Ansh Chhadva
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
chhadva.ansh@fcrit.ac.in

Omkar Bhabal
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
bhabal.omkar@fcrit.ac.in

Lakshmi Gadhikar
Department of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
lakshmi.gadhikar@fcrit.ac.in

Abstract*—According to recent studies, a large number of data scientists have spent almost 60% of their time on tasks like data cleaning and organizing the data. Whereas the entire job of data scientists' includes tasks like data cleaning, performing analysis, modeling, and creating creative data visualizations [1]. Spending almost 60% of their time on trivial but extremely important tasks such as cleaning and organizing is very time-consuming. They also need to remember long complex syntaxes for some of the major tasks in data science which deem to be redundant and time-consuming. So, we propose to build a smart system that enables data scientists to perform all the tedious and time-consuming tasks such as EDA (exploratory data analysis), data cleaning, data pre-processing, data visualization, modeling, etc in the data-science life cycle, by only conveying the logic of the task in natural language and the system will automatically give out relevant python code. Existing applications involving text-to-code generation and code-search are very limited and most do not work in non-ideal conditions. The main reason for this is the dataset that the existing models are based on. These datasets do not account for real-world factors like slang language, acronyms, paraphrased sentences, etc. Therefore, a new dataset was created consisting of real-world user queries which represent the scenarios users are most likely to face during daily usage of this system. Furthermore, using the same dataset we train various intent classification including GloVe based word embedding, Facebook InferSent, Semantic Subword Hashing, and TF-IDF model to find the best technique to classify the intent of user query from our dataset and NER (Named Entity Recognition) model, to identify all the entities present in the sentence. Hence, we plan to build a logic-oriented system that only requires the user to simply convey the logic correctly in natural language via text. The data scientists are not required to remember long syntaxes anymore and the generated code would be very efficient and would follow all the best practices in data science, which will most certainly save time exponentially and they can devote most of their time to building logic.

***Keywords* – text-to-code, data-science, intent-classification, NER, natural-language-to-code, code-generation, code-search**

I. INTRODUCTION

Data Scientist is the sexiest job of the 21st Century. According to LinkedIn's 2017 U.S. Emerging Jobs Report [2], Machine Learning Engineer jobs grew almost 10 fold since 2012, and Data Scientist jobs grew 6.5 times [3]. However, finding qualified people who can work smarter, faster, and can write efficient code to fill such jobs remains difficult. The ability to take data, understand it, process it, visualize it, analyze it and draw meaningful conclusions is going to be a hugely important skill in the next decades. Hence there should be a system that can assist the people doing such complicated tasks.

Our project Codify is to get relevant code snippets from natural language descriptions, for Data Science Applications. This system internally uses an AI system. The term AI that is Artificial Intelligence generally refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. In the view of our project, Codify stands for a smart intelligent system that can code like a human being for a data science application. The user just needs to type what they want in the form of a natural language query (English), and our system will automatically give out the relevant code for it.

There are in general two main techniques that are typically used to get the required code from the given user query in natural language.

1. Code Generation using Encode and Decoder models
2. Code Search Engine

Both of the techniques utilize open-source code snippets from platforms like github, stack overflow, etc. The applications involving text-to-code generation are very limited and most of them do not work in realistic conditions. The main reason is the dataset that these existing models are based on. These datasets do not account for real-world factors like slang language, acronyms, paraphrased sentences, etc. in the input text since they are trained on docstrings obtained from open-source code snippets. Docstrings do help to a certain extent, however, beyond that, they often include summarized versions which do not contain code keywords. These make the datasets unauthentic. Therefore, a new dataset was created consisting of real-world user queries which represent the scenarios users are most likely to face during daily usage of this system.

The existing system, like in the case for code search-based application, the reply maybe is not as per user requirement, so he has to look up to multiple results for finding the code as per his requirement if he doesn't find that particular code

snippet he has to stitch the code by himself, by taking some part of other result code snippet. Also, most of the existing systems are quite large and hence require large computational power to even run the load the model into memory and perform predictions via it. This urges the need for a lightweight solution. Therefore to overcome all these limitations we have come up with our system. In our system, the input given by the user is first cleansed. It is then passed for intent classification to classify the category of user input statement, which is then followed by entity recognition to identify specifically which technique, method, algorithm, approach, etc is to be used. Lastly based on the classified intents and entities, the relevant code is fetched from the code database and made visible to the user. This helps the user to get the complete code snippet instead of just getting either partial code or incomplete functions. This not only makes our system reliable but also fast and efficient. This lightweight system also enables quick prototyping of the model machine learning applications.

II. LITERATURE REVIEW

Supervised Learning of Universal Sentence Representations from Natural Language Inference Data: In 2017, Facebook introduced InferSent as a sentence representation model trained using the supervised data of the Stanford Natural Language Inference datasets (SNLI). SNLI is a dataset of 570k English sentences and each sentence is a pair sentence of the premise, hypothesis labeled in one of the following categories: entailment, contradiction, or neutral. This paper compares sentence embeddings trained on various supervised tasks and shows that the sentence embeddings generated from models trained on a natural language inference (NLI) task reach the best results in terms of transfer accuracy. It also hypothesizes that the suitability of NLI as a training task is caused by the fact that it is a high-level understanding task that involves reasoning about the semantic relationships within sentences. The experiments show that an encoder based on a bi-directional LSTM architecture with max pooling, trained on the Stanford Natural Language Inference (SNLI) dataset, yields state-of-the-art sentence embeddings compared to all existing alternative unsupervised approaches like SkipThought or FastSent while being much faster to train [4].

Subword Semantic Hashing for Intent Classification on Small Datasets: This paper introduces the use of Semantic Hashing as embedding for the task of Intent Classification. With Semantic Hashing, they overcome spelling error and vocabulary challenges and achieve state-of-the-art results on three datasets: Chatbot, Ask Ubuntu, and Web Application. Their process consists of subword semantic hashing followed by preprocessing and data-augmentation, vectorization, intent classification, and evaluation. Their method extracts sub-word tokens from the sentences as features which are then vectorized before they are processed by a classifier for prediction or training. Their method alone can be viewed as a feature and can be complemented with a vectorizer for using it as an alternative to embeddings. The experiments in their paper have been carried out with a number of classifiers, namely K-Nearest Neighbors (KNN) Classifier, Ridge Classifier, Passive Aggressive Classifier, Multilayer Perceptron, Linear Support Vector Classifier (SVC), Nearest Centroid, Multinomial Naive Bayes (NB), Random Forest Classifier, Stochastic Gradient Descent (SGD) Classifier, K-means, Bernoulli Naive Bayes [5].

Citation Intent Classification Using Word Embedding: For citation intent analysis, the datasets must have a citation context labeled with different citation intent classes. But most of them either do not have labeled context sentences, or the sample is too small to be generalized. This leads to their proposal of an automated citation intent technique to label the citation context with citation intent. The annotated ten million citation contexts with citation intent from the Citation Context Dataset (C2D) dataset with the help of their proposed method. They applied Global Vectors (GloVe), Infersent, and Bidirectional Encoder Representations from Transformers (BERT) word embedding methods and compared their Precision, Recall, and F1 measures. The BERT embedding had an 89% Precision score. Finally, It can be used as a benchmark dataset for finding the citation motivation and function from in-text citations. Furthermore, for proposing a text clustering-based mechanism to annotate an un-annotated dataset using the citation context, they proposed a method and evaluated it on the pre-annotated dataset, Sci-Cite. It was observed that contextual embedding played an essential role in grouping the citation sentences, as they provided better results than non-contextual embedding [6].

III. PROPOSED SYSTEM

The proposed system plans to develop a text to code system, in which when a natural language query is given as input it will give out its relevant python code snippets. We achieve this by first collecting our dataset according to our requirements, which is then followed by its annotation. Later we train various intent classification models and the NER model upon it. Now we will look at each of the processes in detail.

A. Dataset collection and Annotation process

Applications involving text-to-code generation are very limited and most do not work in less ideal conditions. The main reason for this is the dataset that the existing models are based on. These datasets do not account for real-world factors like slang language, acronyms, paraphrased sentences, etc. In the input text since they are trained on docstrings obtained from open-source code snippets. Docstrings do help to a certain extent, however, beyond that,

they often include summarized versions which do not contain code keywords. Furthermore, they include a lot of irrelevant text with barely a slight resemblance to the code snippet. These docstrings are more than often in an un-uniformed format and include function parameter definitions, etc. The above-mentioned factors highly disturbed the authenticity of a dataset. Therefore, a new dataset was created consisting of real-world user queries which represent the scenarios users are most likely to face during daily usage of this system.

The Complete dataset is divided into two parts and their format are explained below:

1. **User Query Dataset:** Contains user query and their respective intent. This dataset contains real-world user queries which are collected from many users via survey, in which users were asked to frame queries to get the code that is commonly used in the data-science life cycle, and were also asked to paraphrase some of data science-related queries. Additionally, we have also curated data from Kaggle notebooks, GitHub open-source code, StackOverflow, blogs, and documentation of data-science libraries. Later we manually annotated each user query to their respective intent.

Example: Below is an example from the User Query dataset where the text column contains user query and intent and the entities column contains its specified intent and entities respectively which are manually annotated.

id	text	intent	entities
1	replace null values with mean values	null_imputation	{"entities": [[25, 29, "STATISTICS"]]}

Table 1. User Query Dataset

Dataset Attribute:

- id: Represents unique row number.
- text: It contains an example of a natural language query that would be given by the user.
- intent: For unique identification of each natural language query, this specifies the category under which user query lies like null_imputation, encoding, classification, etc
- entities: Terms that represent methods, techniques, type, etc

From the user query dataset, various intent classification model and NER model is trained.

2. **Codify Dataset:** Contains code snippets, and their identified respective intents and entities.

Example: Below is an example from the Codify dataset where we have python_code for specified intent and entities.

code_id	intent	entity_name	entity_value	python_code
100	null_imputation	STATISTICS	mean	df=df.fillna(df.mean())

Table 2.Codify Dataset

Codify Dataset Attribute:

- code_id: Represents unique code snipped ID.
- intent: For unique identification of each natural language query, this specifies the category under which user query lies like null_imputation, encoding, classification, etc
- entity_name: Terms that represent methods, techniques, type, etc
- entity_value: Value of entity obtained from user query like method/technique name, etc
- python_code: Contains relevant code of corresponding text in the text column.

After identification of intent and entities from user query, its respective code is fetched from this dataset and given as output.

Parameters	Statistics
Total Number of Users Queries	525
Total Number of Unique Intents	20
Total Number of Unique Entity	10
Total Number of Unique Python Code Snippets	100

Table 3. Dataset Stats

The complete dataset consists of user queries, intents, entities, and their respective python code snippets. There are a total of 525 user queries collected which are classified into 20 intents, and then if any entity is present in that query we annotate it from one of the 10 unique entities. And finally, we have a total of 100 unique python code snippets which are given as output.

B. Codify Architecture

In our system, the input given by the user is first cleansed. It is then passed for intent classification to classify the category of input user statement, which is then followed by entity recognition to identify specifically which technique, method, algorithm, approach, etc is to be used. Lastly based on the classified intents and entities, the relevant code is fetched from the code database and made visible to the user.

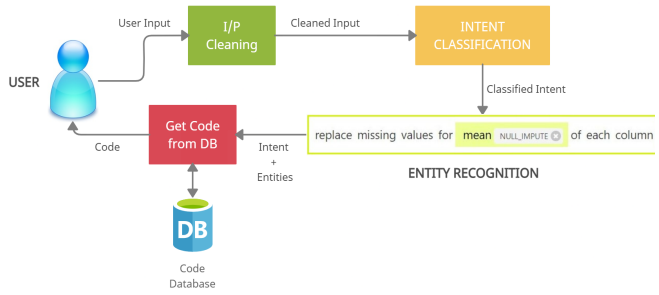

Fig. 1. Codify Architecture Diagram

a) Input Cleaning: The input given by the user is first cleansed by making the sentence free from all punctuation marks, quotations, leading, and trailing extra spaces. Also, the acronyms present in the sentence are replaced with their respective full form and every word is present in lowercase format.

b) Intent Classification: The cleaned input is then passed to the best intent classification model which will classify the intent of the given user query.

c) Entity Recognition: After intent classification, we pass our textual data through the Named Entity Recognition (NER) model. Named Entity Recognition is the process of extracting predefined entities like the name of a person, location, organization, time, date, etc. from unstructured text data. NER assigns a named entity tag to a designated word by using rules and heuristics.

d) Get Code from DB: From the classified intent and the recognized entity of the given user query, we fetch the relevant code from the "Codify Dataset", format it accordingly, and send it to the user.

C. Training and Testing

Sr.no.	Method	Accuracy
1.	Sum of Word Embedding using GloVe	88.60%
2.	Facebook InferSent	87.34%
3.	Semantic Subword Hashing	78.48%
4.	TF-IDF	74.68%

Table 4. Intent Classification Techniques Accuracy

The complete dataset was divided into 85% train set and 15% test, giving us the accuracy as shown in table 4. The best result was achieved by the "Sum of Word Embedding using GloVe" technique giving 88.60% accuracy. It was then tested with another effective approach, Facebook InferSent, which is an encoder-based bi-directional LSTM architecture with max pooling. Trained on the Stanford Natural Language Inference (SNLI) dataset, it yielded an accuracy of 87.34%.

Semantic Subword Hashing was the next process we trained the model on our dataset. It is a process consisting of subword semantic hashing which is followed by preprocessing and data-augmentation, vectorization, intent classification, and evaluation. It gave an accuracy of 78.48%. Finally, we used the TF-IDF approach where the first task was to pre-process the user queries, and then the context vectors were created using word embedding techniques. In the final stages, we created clusters of the word embeddings, and then after studying those created cluster samples, we assigned an intent to each of these clusters. This method of intent classification gave a pretty decent accuracy of 74.68%.

D. Sum of Word Embedding using GloVe

As you can observe, we tried a lot of ways to classify intent pertaining to our dataset, out of which we found the "Sum of Word Embedding via GloVe" approach to be the best. The basic idea behind the GloVe word embedding is to derive the relationship between the words from statistics. Unlike the occurrence matrix, the co-occurrence matrix tells you how often a particular word pair occurs together. Each value in the co-occurrence matrix represents a pair of words occurring together.

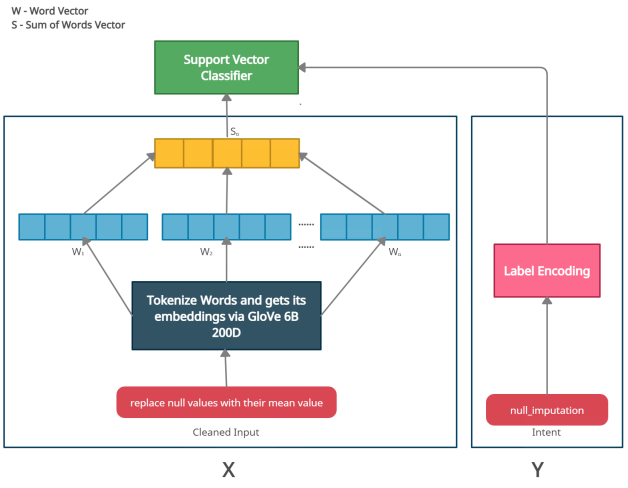

Fig. 2. Sum of Word Embedding using GloVe Architecture

As you can observe in the diagram above we tokenized each word from the sentence then used GloVe embeddings of 6 billion tokens and 200 dimensions, to get the word embeddings of each word. Then we sum them up to get the sentence embeddings. And later this sentence embedding is fed into the SVC (Support Vector Classifier) model which will classify intent.

VI. IMPLEMENTATION

We have implemented it in two ways i.e API implementation and Live code editor implementation.

A. API implementation

For the demonstration of API, we have built a website where the user writes his query in the user input box, after writing he clicks on the button “Get Code”. When the button gets clicked the query is passed to the server via the post request. In the server, the user query gets cleaned and preprocessed. After which by identifying its intent and entities, we fetch all the relevant code snippets from Codify dataset and give it as a response to the user.

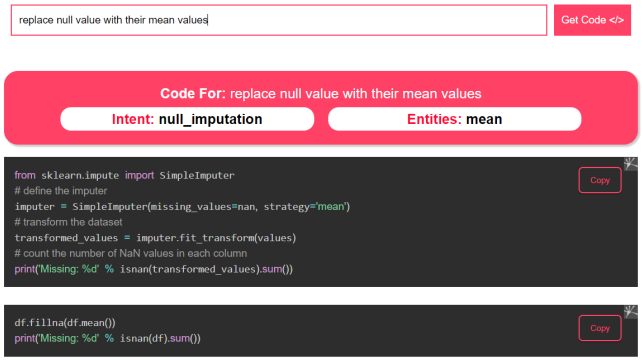

Fig. 3. Codify API Example

Example: The user types his query in the input box “replace null value with their mean values” and clicks on the “Get Code </>” button. After which the following code card is displayed as shown in figure 3.

B. Live code editor

In Live code editor implementation, when the user writes code in the code editor, he activates our system by writing his queries in python comment format, that comment is then passed to the server via the post request. In the server, the user query gets cleaned and preprocessed. After which by identifying its intent and entities, we fetch all the relevant code snippets from Codify codebase and give it as a response to the user.

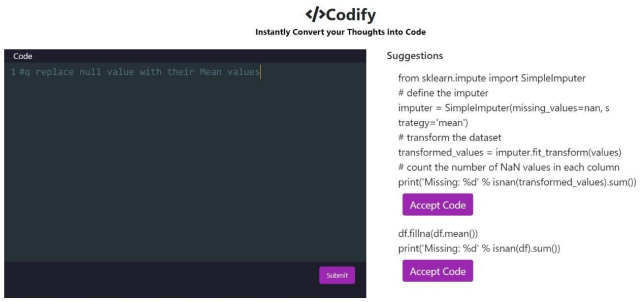

Fig. 4. Codify Live Code Editor Example

The user type comment starting with “#q” to activate our system. As the user finishes the sentence and hits the ENTER key. The query is sent to the server and the server responds with code suggestions as shown in figure 4.2. Then the user clicks on the "Accept Code" button for 1st snippet, After which those particular code snippets get pasted in the code editor where the user cursor is blinking as shown in figure 4.

Fig, 5. Codify Live Code Editor Example with Code Output

V. CONCLUSION

We have presented our system, Codify, with the primary aim of easing the Data Science related tasks which will save time and he/she can refrain from remembering and writing long complex syntax which is often redundant in the data-science lifecycle. Our system is capable of taking the input in the text or speech format in Natural Language and converting them to its relevant python code. We have built Codify system using the dataset consisting of real-world user queries written in natural language, their respective intent, and entities, and lastly its relevant corresponding python code. The user queries are fed into the intent classification model and entity recognition model to identify the intent and all the various entities present in the given user query, after which using the identified intents and entities we fetch the corresponding python code from the code database. To further reduce writing redundant, long, and time-consuming code.

ACKNOWLEDGMENT

We would like to express our deepest appreciation and profound respect to all those who guided and inspired us in the completion of our project ‘Codify: "Natural Language to Code, for Data Science Applications" ‘.We would also like to thank our guide Prof. Lakshmi Ghadikar guided us throughout in achieving the goal. Our sincere gratitude to Dr. S. M. Khot, Principal of Fr. C. Rodrigues Institute of Technology, and HOD Ms. Dhanashree Hadsul for providing us the opportunity to implement. Furthermore, I mentioned our dedicated team members and friends who were always rooting for us to successfully complete this paperwork.

REFERENCES

[1]http://www2.cs.uh.edu/~ceick/UDM/CFDS16.pdf [Accessed: May 5th, 2021]

[2]https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century. [Accessed: May 5th, 2021]

[3]https://economicgraph.linkedin.com/research/LinkedIns-2017-US-Emerging-Jobs-Report. [Accessed: May 5th, 2021]

[4]Alexis Conneau, Douwe Kiela, Holger Schwenk, Lo¨ıc Barrault, Antoine Bordes “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”.

[5]Kumar Shridhar, Ayushman Dash, Amit Sahu, Gustav Grund Pihlgren, Pedro Alonso, Vinaychandran Pondenkandath, Gy ¨orgy Kov´acs, Foteini Simistira, Marcus Liwicki “Subword Semantic Hashing for Intent Classification on Small Datasets”.

[6]Muhammad Roman, Abdul Shahid, Shafiullah Khan, Anis Koubaa, And Lisu Yu “Citation Intent Classification Using Word Embedding”.