STACK OVERFLOW LIVE DATA VISUALIZATION & ANALYSIS

Aatir Fayyaz
Department of Mechanical Engineering
Columbia University In the City of New York
af3252@columbia.edu

Kunlun Wang
Department of Biomedical Engineering
Columbia University In the City of New York
kw2964@columbia.edu

Muhammad Sajawal Saghir
Department of Mechanical Engineering
Columbia University In the City of New York
ms6339@columbia.edu

I. Abstract

The main goal of this project is to visualize real time StackOverflow tags and title information to visualize, analyze and understand the changing programming trends and challenges faced amongst software users around the world. Experiments were implemented on the data obtained using the StackExchange API and several visualizations were produced including an interactive bubble (or circle packing) chart, a bar chart race, and a pie chart. These live visualizations would help understand the popularity of a particular programming language so users can understand current popular languages, and make valid inferences about the future of programming & potential software growth.

Keywords: StackOverflow, Spark, Hadoop, D3, google-cloud, Machine Learning, Visualization, Bubble Chart, Bar Chart Race, Pie Chart.

II. Introduction

Stack Overflow is one of the most popular and engaging question & answer platforms for programmers. It features questions and answers on a wide range of topics in computer programming with feedback through votes, tags and ratings for questions. With a rise in popularity of programming languages, programmers of all levels interact with Stack Overflow for their concerns and questions. Stack Overflow collects a variety of data including questions, answers, number of votes and tags. This makes it a very exciting source of information to build an analysis on to determine knowledge about technologies and languages just by looking at the tags data. Figure 1 shows a typical question graphical user interface on StackOverflow.

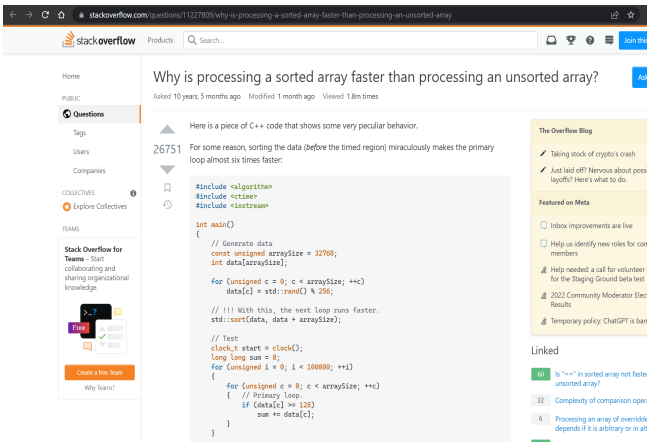


Figure 1: A typical question on Stack Overflow

Each question posed on the Stack Overflow website is associated with a number of tags. A tag as defined by Stack Exchange "is a word or phrase that describes the topic of the question". Tags allow a question to be answered promptly since superusers for a particular tag follow any activity that is linked to it. Tags are also useful in helping a user identify any questions that may be relevant to them.

Figure 2: Tags associated with the question in Figure 1

The purpose of this project is to fetch real time StackOverflow tag data in order to visualize, analyze and understand the rapidly evolving programming trends amongst users around the world. Furthermore, the analysis would also involve a machine learning model prediction to show if the quality of the question being posed is high or low and whether it requires an admin to intervene in order to make the question quality better. This project utilizes the Stack Exchange API which allows users to retrieve and

view information used by the Stack Overflow. As a regular user, Stack Exchange API limits data requests to approximately 300 calls per day. It is recommended to register for an API Key with Stack Exchange which increases that limit to 10,000 calls per day.

Stack API is a python wrapper for the Stack Exchange API. Stack API was used as the data transmitter along with PySpark streaming receiver to fetch live tags from the questions being posed on the Stack Overflow website with an interval of 10 seconds. Data received by the PySpark streaming socket was stored in Google BigQuery after some processing.

III. Related Work

Work has been done in the past with Stack Overflow data dumps of multiple years running statistical data analysis on user activity and generating graphs on matplotlib to show trends of registered users, number of asked questions, number of answers and comments. [1]

Some other projects include creating a curriculum and a learning management system for Stack Overflow which acts as an offline dataset providing essential encyclopedias for programmers without an internet connection. This was accomplished using Stack Overflow data dumps for multiple years benefitting the communities. Work is currently being done to ensure that this database is constantly updated to provide an up to date version of data set to those who need it while also improving readability so there is less friction for end users. [2]

Finally, some big data analyses have been done on the stack overflow data exploring characteristics of users, questions answering pyramid and topical trends in the tags. Analysis was done based on the user behavior classifying them in four categories namely questioner, answerer, question and answerer and do nothing. With more than 50,000 tags in the StackOverflow tag based behavior analysis has been implemented to gauge what the developers care about and visualizations include what tags get more attention compared to the others. [3]

All related works dealt with historical datasets and visualized mostly by matplotlib in python. This project focuses on real-time data acquisition as well as live visualizations and live ML-based question quality prediction.

IV. Data

A. Data Procurement and Pre-Processing

The Stack Exchange API enables users to retrieve answers, comments, badges, events, questions, revisions, suggested edits, user information, and tags from a Stack Exchange based website. It uses RESTful API architecture to make requests and receive responses in a JSON format. Streaming data was pulled from StackAPI every 10 seconds. PySpark streaming was utilized to receive data from Stack API which was sent over the local socket. The data received from the socket was captured in a datastream and for each Resilient Distributed Dataset (RDD), data preprocessing was done to split each tag and remove any unwanted characters. The data that was sent over the socket was also processed such that from the data packet received in a fetch call - only the tags, the title, and the body of each question were transmitted.

In particular, the fetch call on StackAPI builds the API query for Stack Exchange which in turn processes the request. The return type is a python dictionary containing all the data for the requested call including call and client specifics. The relevant data for the query sits under the items list. A typical dictionary is shown in Figure 3 below.

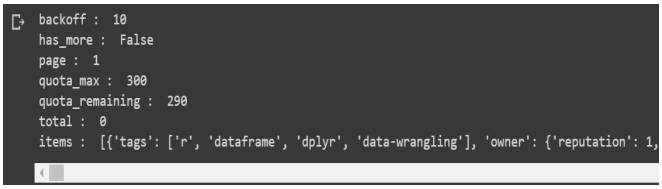


Figure 3: A typical response for a fetch request

Traversing the items list leads to the required data for each of the project's requirements. This includes tags, title, and the body of each of the incoming questions. Since the request for the call is configured to fetch ten (10) seconds worth of data every 10 seconds, each request may contain multiple items lists. It was configured in the code to iterate through these and scrape all relevant data which was then sent to the socket as an encoded, and delimited string.

As seen in Figure 4 below, all the relevant data is available in the items list - this processed data is sent over to the PySpark streaming receiver.

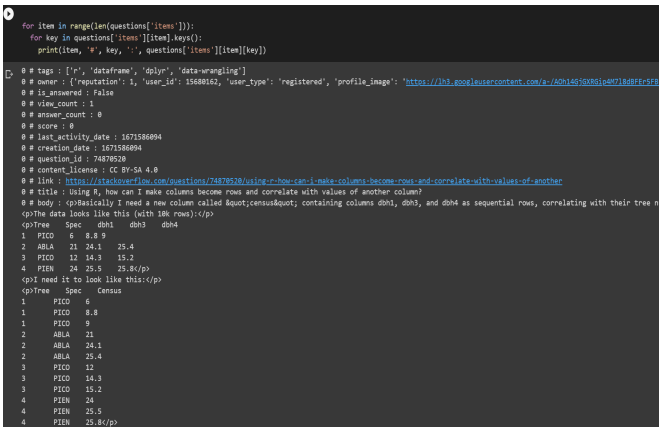


Figure 4: A typical items list

B. Methods

This project utilized Cloud Dataproc on google cloud to run the cluster which fetched and stored Stack Overflow site data using PySpark streaming, Google Cloud Storage and Google BigQuery. As mentioned previously, Stack Exchange API was used to fetch the data. PySpark was utilized for data filtration and MapReduce operations. Pandas-gbq was utilized for storage to Google BigQuery. The MapReduce operation was used to calculate the count of each tag per streaming window which was set at 10 seconds each (since that was the duration for our data request as well). All data received was then stored in BigQuery so that it would be readily available for use by the deep learning model for prediction. Two different types of data tables were stored, one for the tag count application and another for the ML-based question quality analysis application.

Once the data was procured from Stack API and available in BigQuery, data was then imported using Google OAuth2.0 authentication with Google BigQuery. For the purposes of this project, a simple python script was developed to pull the BigQuery data and save it as a .csv file. The pandas and pandas-gbq package was utilized to make this process seamless. Visualizations were completed in javascript using D3.js.

D3 is a javascript (hence, '.js') library which allows the manipulation of an Hypertext Markup Language (HTML) document by binding any data to a Document Object Model (DOM), and then utilizing transformations based on that data to the HTML DOMs. D3 efficiently manipulates web standards like SVG and CSS with minimal overload. It is dynamic in interaction and animation with easily available and reusable code repositories. Therefore, D3 essentially allows for the efficient manipulation of documents based on data and does not need any other packages for the visualizations granting for an overall open source setup.

Current visualizations include an interactive circle packing / bubble chart and a bar-chart race to visualize the incoming tags in real-time.

C. System

The system architecture overview is shown in Figure 5.

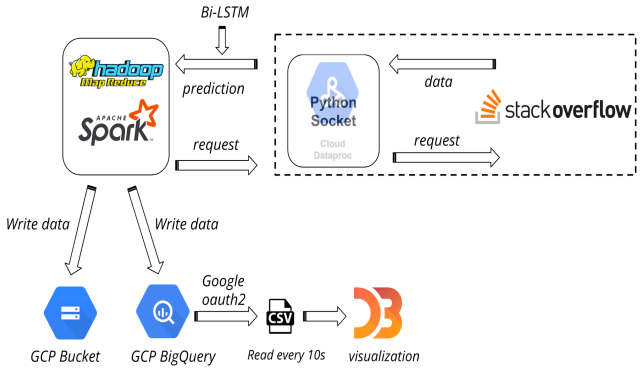


Figure 5: System Architecture

As shown in the figure above, the Stack API requests data from Stack Exchange. Specifically, the tags, question title, and the question body from the latest questions on StackOverflow were procured. Stack Exchange gets the request and sends the requested data to the requester, which in the case of this project is the python socket resting in a Google Cloud Platform (GCP) Dataproc cluster. Within the socket, a Bi-LSTM model trained on Kaggle question quality dataset, with 84% accuracy, is used to make prediction of the question quality of the live data. Finally, the data is then transmitted through the local socket (locally hosted over socket 9001) to the PySpark streaming receiver service deployed on dataproc, which accepts the handshake from the local socket.

Within the PySpark streaming receiver, incoming data including the tags, question title, and question body is read as a datastream. A MapReduce Operation is used to count the number of each tag, and question title, body, and prediction is processed into a dataframes and a screenshot of them is stored into the Google Cloud Bucket temporarily until it is pulled by pandas-gbq and stored to Google BigQuery. A python pandas-gbq instance with Google OAuth2.0 for BigQuery access is able to access this data remotely and stores the accessed data as a .csv file locally.

The locally saved comma separated values (.csv) file is then read by d3.js in javascript with the d3.csv() function. A

HTML-based web application is constructed incorporating the javascript code and CSS styles to finalize the visualization pipeline.

V. Experiments

Once the data is retrieved from Google BigQuery, the d3.csv() function is used to parse this data into relevant variables. d3.selectAll() and d3.enter() use these variables to create required DOM elements and manipulate them using the Support Vector Graphics (SVG) in HTML. Since the team was dealing with a live datastream, the d3.setInterval() function was used to update the visualization every ten seconds.

One of the challenges that the team faced was the continuously updating .csv file. An update to a .csv file resulted in a complete memory flush for the javascript variables. This meant that the variables lost any old data that they contained and that was not acceptable for our visualizations since it required the entire datasheet. Since passing data outside the d3.csv() function and similar .csv file reading functions (like jquery) were found to be impossible due to the asynchronous nature of javascript calls as well as due to security concerns in browser-based applications - this posed a large roadblock to our live visualization development.

To remedy this, when saving to Google BigQuery, the data was appended to the existing data rather than replaced completely with each session starting with an empty data table so all data captured was still live from the time of inception. This is a workaround that the team implemented but future work is required to fine tune the method to make it computationally and resource efficient since this is not the most optimal solution.

A. Live Circle Packing Visualization

Circle packing refers to arranging the circles of the same or different sizes such that no overlapping of the circles occurs. Circle packing in the context of visualization is completed using a value as a determinant for the size of the circle. In the data procured by the team, this refers to the count of the tags over time. This type of visualization is also referred to as a bubble chart. [4][5][13][14]

Since the .csv file saved by the dataproc cluster ran a MapReduce per RDD, that meant that there were various occurrences of the same tag occurring over multiple streaming windows. Therefore, a function in javascript was deemed necessary which would save the total count of a particular tag so that the visualization did not contain multiple instances of the same tag but rather a sum of all the counts of a reoccurring tag.

Figure 6 and Figure 7 show the circle packing visualization at two different points in time of the data procurement. As seen below, the circles start off small but the popular tags quickly start to gain more and more traction and are seen to grow in size.

To make the circle packing graph further interactive and user friendly, d3-force was implemented to the visualization. D3's force layout, essentially, incorporates a physics based simulator to position the visual elements on the SVG canvas. The implementation allowed the movement of the circles by clicking and dragging any circle around the visualization. Figure 8 below shows a screenshot of dragging one of the nodes around the canvas.

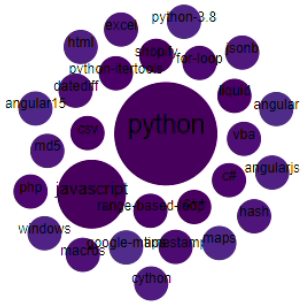


Figure 6: Circle Packing Chart (~1 min into data gathering)

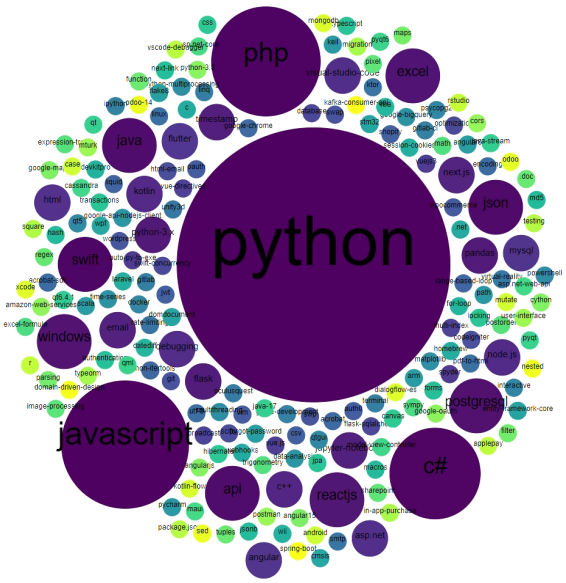


Figure 7: Circle Packing Chart (~10 min into data gathering)

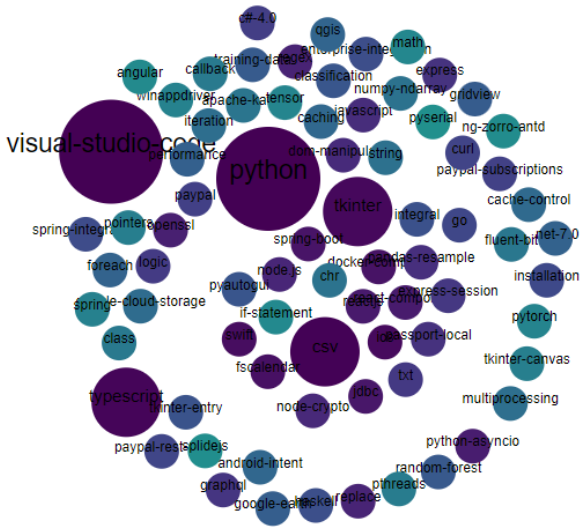


Figure 8: A frame of Circle Packing with Force Simulation

From the visualizations in Figures 6 and 7 above, it can be seen that python, javascript, php and c# are the most popular and sought after tags amongst the plethora of other tags for that particular time period. An interesting fact that was noticed during the development of these charts is that python and javascript definitely reigned supreme for the current time period (i.e. Dec 2022).

B. Live Bar Chart Race

A bar chart race is a great way to visualize highly changing data over time in the form of an animated bar chart. The chart consists of four parts: the bars, the x-axis, the labels, and the ticker showing the current date. The animation iterates over each of the keyframes, delegating updates to each of the four chart components and awaiting the transition's end. On each step series position delta is calculated and set an animation on the data-item so when position changes it smoothes into an animation rather than jumping into a new place. [6][15][16]

Similar to the circle packing chart, the bar chart race also required similar functions to append to the already existing count. Figure 9 below shows the visualization of the bar chart race.

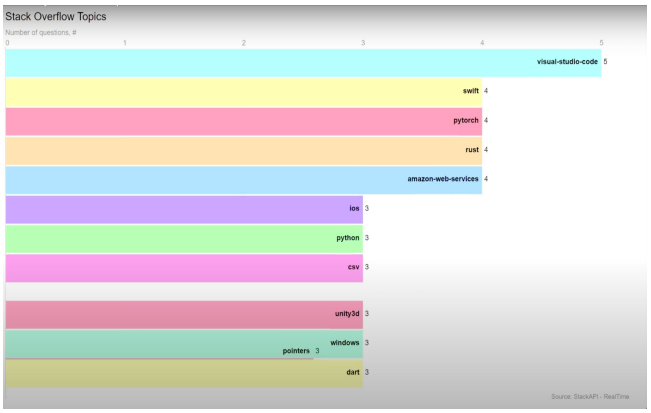


Figure 9: Bar Chart Race Visualization

Figure 10 : Bar Chart Race Visualization around midnight

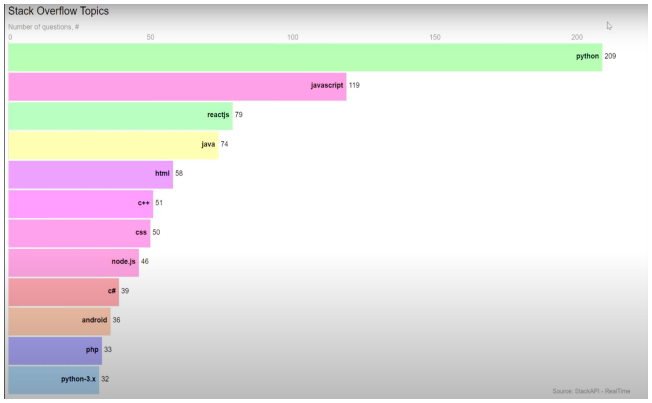


Figure 11 : Bar Chart Race Visualization around daytime

In order for a better visualization, we fetched the data overnight to see how the trend changes as the night progressed. Figure 10 and Figure 11 above show the bar chart race visualization at two different points in time of the data procurement. It can be observed that there is a stark difference in terms of the programming language questions being asked. Around midnight swift and visual studio code were the most popular tags while during the daytime python and javascript were the most popular tags. This shows how the user preference changes during different times of the day

probably due to the active times in the different countries around the world. So in essence, it highlights the favorable programming languages in different countries.

C. ML-Based Application

To get a better understanding of the quality of the questions being posted in real-time, we employed a Bi-LSTM model trained on Kaggle's "60k Stack Overflow Questions with Quality Rating" dataset to classify incoming questions into three categories: HQ, LQ_EDIT, and LQ_CLOSE. We achieved a weighted accuracy of 84%.

i) Dataset

Kaggle's *"60k Stack Overflow Questions with Quality Rating"* dataset, which contains 60,000 question samples collected from the Stack Overflow website from 2016 to 2020, was utilized for the data analysis. The dataset consists of the unique user IDs, question titles, the body of the questions (content), tags, the creation date of the question, and the label for each question. [7]

The label consists of three classes:

1. High-Quality (HQ): Questions that received a score of more than 30 from the community and did not need to be edited.
2. Low-Quality-Edited (LQ_EDIT): Questions that received a negative score and required one or more edits from the community.
3. Low-Quality-Closed (LQ_CLOSE): Questions that were closed by the community within a short period of time due to their poor quality.

The dataset is very well-balanced, each class has exactly 20,000 samples as it shows in Figure 12, this is potentially advantageous for our training as it prevents the model from being biased towards a certain class.

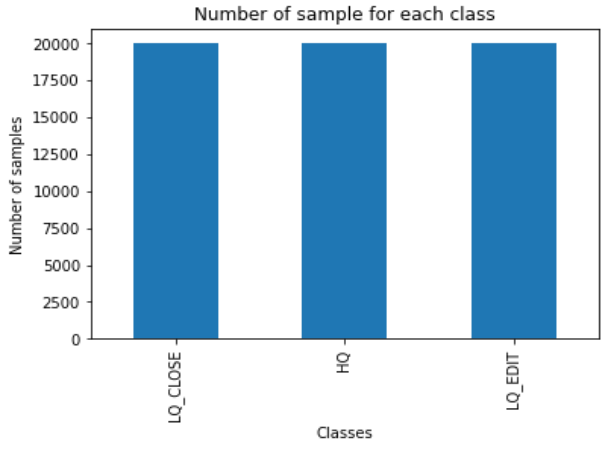
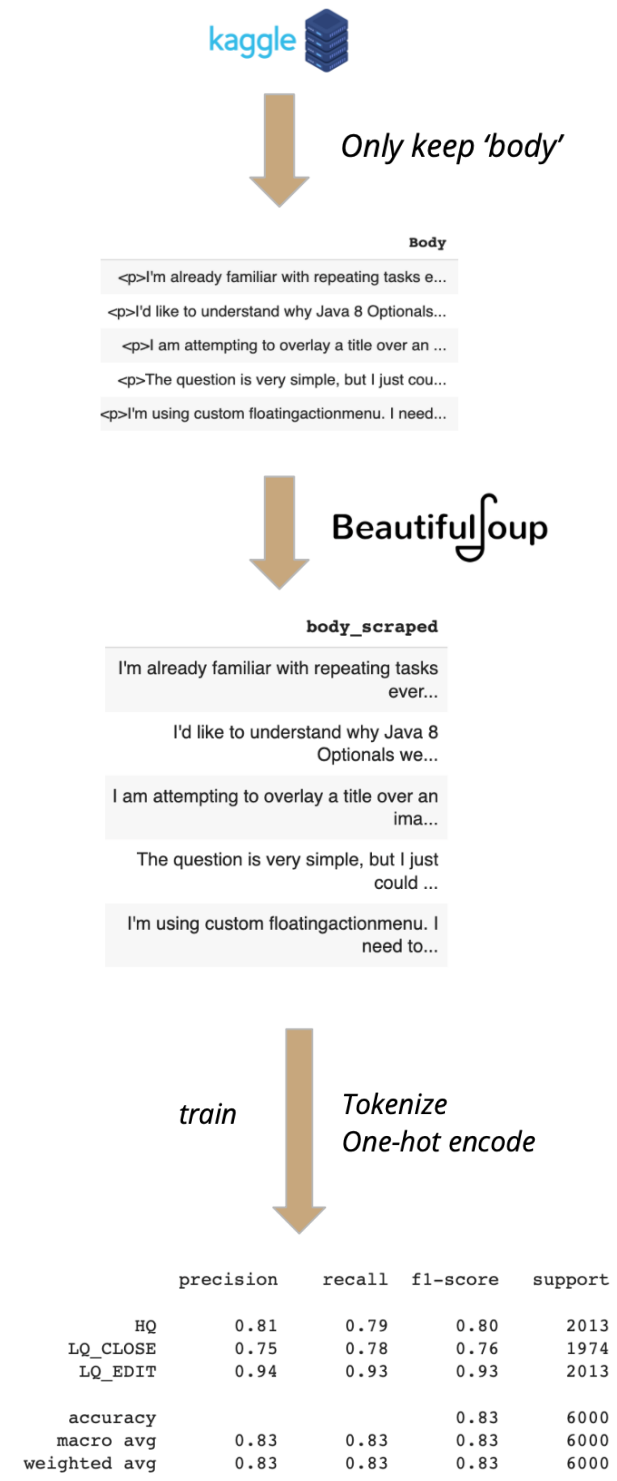


Figure 12: Sample size for each class for LSTM Model



	precision	recall	f1-score	support
HQ	0.81	0.79	0.80	2013
LQ_CLOSE	0.75	0.78	0.76	1974
LQ_EDIT	0.94	0.93	0.93	2013
accuracy			0.83	6000
macro avg	0.83	0.83	0.83	6000
weighted avg	0.83	0.83	0.83	6000

Figure 13: The preprocessing steps of the dataset

ii) Preprocessing

Figure 13 shows the workflow of the preprocessing steps of the dataset. First, the features that were not very relevant to deep learning or those that might impose bias to training were filtered out of the dataset. The features that were

dropped included the unique User IDs, tags, and the creation date of the question. Only the question title and body was kept for the training.

The question body was further filtered using a Python Web Scraping package called BeautifulSoup to remove unwanted HTML annotations. A histogram for all three classes was plotted to see the distribution of the body length, and as it shows on Figure 14, the majority of the data is less than 1000 words long, so we empirically capped the length of the body to be 1000 such that every data sample will have the same length for the model training. The resulting text was tokenized using the Keras tokenizer. The tokenizer was configured with a vocabulary size of 100,000 to ensure the capture of the majority of the words. Any words that were not in the vocabulary were then replaced with an empty string. After the text was tokenized, it was converted into sequences using keras *text_to_sequence,* so that it could be fed to the deep learning models. The processed text was then split into the training, testing, and validation sets to calculate model performance metrics. [11][12]

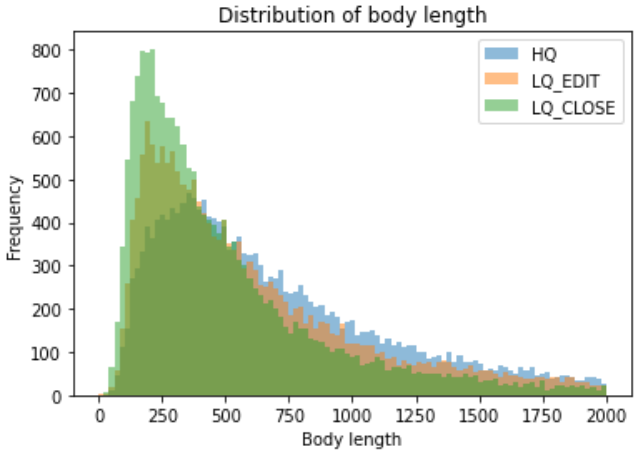


Figure 14: Body Length Distribution for LSTM Model

iii) Modeling

A Bidirectional LSTM (Bi-LSTM) model was implemented for making the prediction of question quality of the incoming data. An LSTM network is a type of recurrent neural network that is capable of learning long-term dependencies in data; it is particularly useful for sequential data because it can remember previous input sequences and use that information to better predict for newer ones. So, LSTMs are renowned for their state of the art performance on tasks such as language translation and text classification, where it is important to consider both the preceding and the following words in order to accurately understand the meaning of a given word. A high-level schematic of an LSTM network is shown in Figure 15. [10]

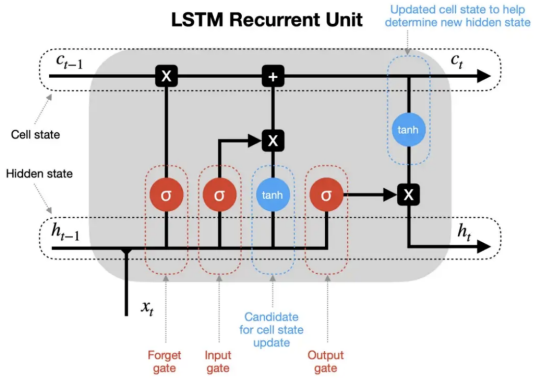


Figure 15: Long Short-Term Memory (LSTM) Neural Networks Architecture

The variation of the LSTM model that was implemented (Bi-LSTM) can take into account both the past and the future text when making the prediction. This is very important because some words might have temporal relationships with either previous or future words in a natural language sequence (like the title or the body of a question on StackOverflow). Bi-LSTMs can achieve this by running the vectorized word sequence through two LSTMs - one that processes the data in the forward direction and another that processes the data in the reverse direction. The outputs from the two LSTMs are concatenated into a single output vector that contains information about both the past and the future context of the input words.

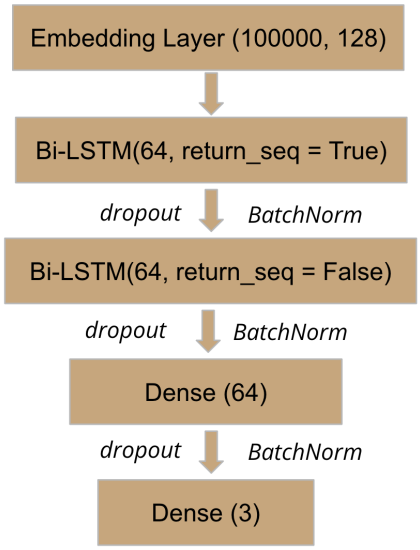


Figure 16: Architecture of the Bi-LSTM model

The overall architecture of the implemented Bi-LSTM can be seen in Figure 16. The Bi-LSTM model consists of two types of layers: 2 Bi-LSTM layers, and 2 dense layers for classification. Each Bi-LSTM layer consists of two LSTM modules, both modules simultaneously process the word sequence temporal information in two opposite directions. As explained earlier, one LSTM module will process the data in forward direction, while the other LSTM module does it in the opposite direction. Each LSTM module will produce its own output, and then the two outputs are combined to determine the output at a specific step. The model, therefore, is able to take into consideration the potential dependencies of the word sequences during classification.

Both Bi-LSTM layers are followed by a batch normalization layer and a dropout layer (with a dropout rate of 0.2) for higher training speed and reduction of overfitting. The LSTM layers are followed by two dense layers that are initialized with he_normal and are activated using ReLU. The final dense layer is activated using the softmax function. The Bi-LSTM model is optimized using the Adam optimizer (with a learning rate), and categorical cross entropy loss function. The Bi-LSTM model was training for 100 epochs with batch size of 32 and an early stopping mechanism based on the validation loss.

iv) Performance analysis

The model achieved an average of 84% accuracy for classifying the questions. In particular, it seemed that the model was very effective at classifying the LQ_EDIT questions, with an overall accuracy of 96%. However, for the HQ questions and LQ_CLOSE questions, the model achieved an accuracy of ~84% and ~75% respectively. The majority of the misclassification happened between HQ and LQ_CLOSE questions, as can be seen in Figures 17 (which shows a detailed classification report) as well as in Figure 18.

	precision	recall	f1-score	support
HQ	0.84	0.78	0.81	2013
LQ_CLOSE	0.75	0.83	0.79	1974
LQ_EDIT	0.96	0.93	0.94	2013
accuracy			0.85	6000
macro avg	0.85	0.85	0.85	6000
weighted avg	0.85	0.85	0.85	6000

Figure 17: Classification report of the Bi-LSTM model

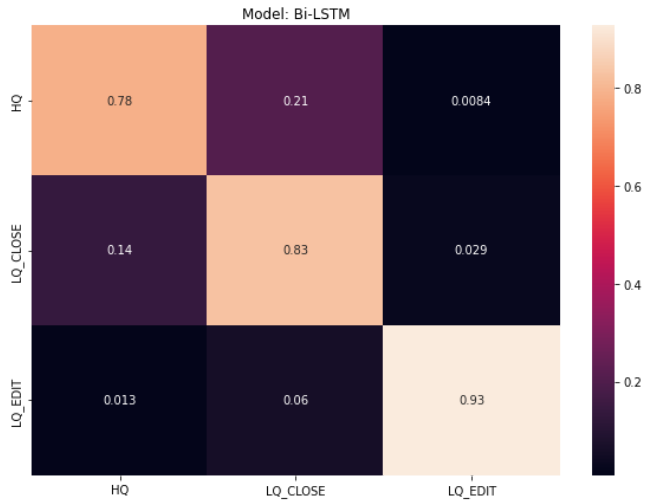


Figure 17: Confusion matrix of the Bi-LSTM model

v) Prediction Visualization

After training, the model weight was saved and inserted into the python socket to make predictions of the incoming questions. The model was configured to take the incoming question title and body, process it and then feed it to the model weights in real-time to generate a predicted question quality. The predictions are saved to BigQuery and are read as a .csv file every 10 seconds through the same Google OAuth2.0. The number of each predicted class is then plotted as a pie chart using D3.js as shown in Figure 19. The file refreshes every 10 seconds to reflect additional data.

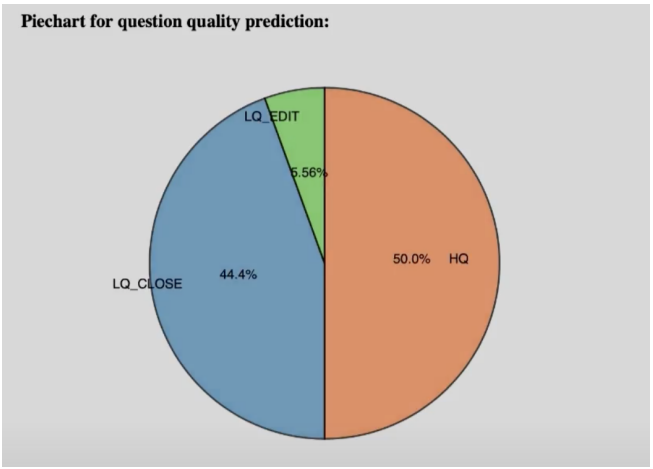


Figure 19: Pie chart of predicted question quality

A table that contains the title and prediction of the incoming questions is also generated, as shown in Figure 20.

Table for title and prediction:

Title	Classification
how to stabilize my socket to not crashing when one side disconnect	LQ_CLOSE
C# Trying to save a picture from a user in a web browser	LQ_CLOSE
Make custom fixed interval chartjs	HQ
Python Selenium opens Website but then Website closes	HQ
How can i get this deal card fuction working for a BlackJack game?	LQ_EDIT
Should I put this code in the shouldUpdate call?	LQ_CLOSE
How to approach setting up domain2 (server2) pointing to domain1 (server1) and creating 2 subdomains on domain2 for 2 apps both using SSL?	HQ

Figure 20: Table showing Title and Predicted Quality of Incoming Questions

vi) Future experiments

To further enhance the validity of the question quality prediction, the team plans on implementing a transformer-based model due to its superior performance in NLP classification. The planned transformer model will follow an architecture of a positional embedding input layer, repeated multi-head attention layers, GlobalAveragePooling layer, and dense layers for classification (as shown in Figure 21). [8]

Each multi-head attention layer will consist of:

1. One multi-head attention mechanism with n-heads, k-key dimension, v-value dimension.
2. One dropout and layer normalization following the multi-head attention.
3. A fully-connected convolutional layer with kernel size = 1.
4. Another dropout and layer normalization following the fully-connected convolutional layer.

The transformer model will be optimized using the Adam optimizer (with a learning rate), and categorical cross entropy loss function. As a start, the transformer model will be trained for 100 epochs with batch size of 32 and an early stopping mechanism based on the validation loss.

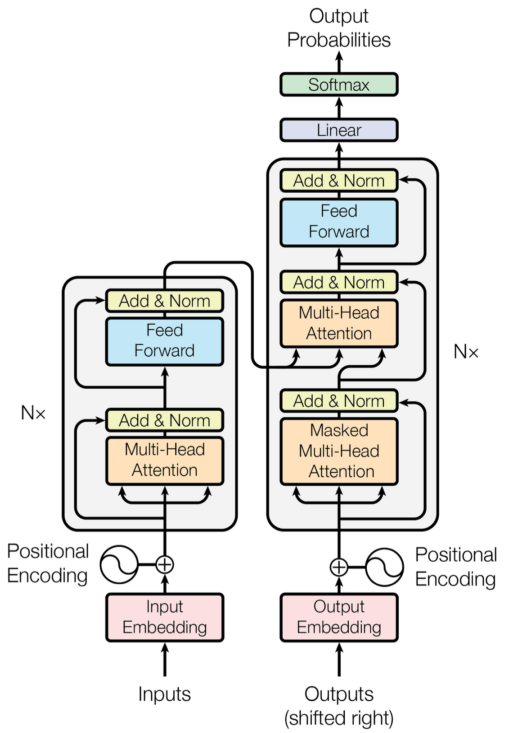


Figure 21: The Transformer Model Architecture

VI. Conclusion

Big data analysis is a necessity nowadays as it has a significant impact and many organizations are starting to realize the importance of big data. It helps organizations harness their data and use it to identify new opportunities which result in smarter business decisions. Individuals and organizations are able to reduce costs, make faster and better decisions and are able to develop and market new products and services which gives them an advantage over their competitors. The main focus of this project is to better analyze the StackOverflow data and output the results using interactive visualizations. It can be observed that daytime trends are different from night time. Python is the most popular programming language during the daytime while visual studio code and swift are popular during night time. The developed project is a successful application for the analysis of StackOverflow data. [9]

References

[1] Hartmann, B. (n.d.). *~Bjoern/projects/stackoverflow*. Stack Overflow Research Project. Retrieved December 5, 2022, from https://people.eecs.berkeley.edu/~bjoern/projects/stackoverflow/

[2] Popper, B. (2022, October 26). *Introducing the overflow offline project*. Stack Overflow Blog. Retrieved December 5, 2022, from https://stackoverflow.blog/2022/10/20/introducing-the-overflow-offline-project/

[3] Liu1204, Tony. (2020, August 18). *Stack overflow big data analysis*. Kaggle. Retrieved December 5, 2022, from https://www.kaggle.com/code/tonyliu1204/stack-overflow-big-data-analysis/notebook#3.TAG-BASED-ANALYSIS

[4] Wikimedia Foundation. (2022, October 27). *Circle packing*. Wikipedia. Retrieved December 5, 2022, from https://en.wikipedia.org/wiki/Circle_packing

[5] Bostock, M. (2021, December 9). *Circle packing*. Observable. Retrieved December 5, 2022, from https://observablehq.com/@d3/pack

[6] Bostock, M. (2021, April 14). *Bar Chart Race*. Observable. Retrieved December 5, 2022, from https://observablehq.com/@d3/bar-chart-race

[7] Annamoradnejad, I., Habibi, J., & Fazli, M. (2022). *Multi-view approach to suggest moderation actions in community question answering sites*. Information Sciences, 600, 144-154.

[8] Cristina, S. (2022, November 15). The Transformer model. MachineLearningMastery.com. Retrieved December 5, 2022, from https://machinelearningmastery.com/the-transformer-model/

[9] "Big Data Analytics: What It Is and Why It Matters." *SAS*, www.sas.com/en_us/insights/analytics/big-data-analytics.html.

[10] Singh, Gourav. "LSTM Architecture: Understanding the LSTM Architecture." *Analytics Vidhya*, 21 Jan. 2021, www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/.

[11] "Beautiful Soup Documentation¶." *Beautiful Soup Documentation - Beautiful Soup 4.9.0 Documentation*, www.crummy.com/software/BeautifulSoup/bs4/doc/.

[12] Real Python. "Beautiful Soup: Build a Web Scraper with Python." *Real Python*, Real Python, 15 Dec. 2022, realpython.com/beautiful-soup-web-scraper-python/.

[13] Bostock, Mike. "Circle Packing, Bubble Chart." *Observable*, 24 Oct. 2021, observablehq.com/@d3/bubble-chart.

[14] "How to Create a Bubble Chart Using D3." *Educative*, www.educative.io/answers/how-to-create-a-bubble-chart-using-d3.

[15] Bostock, Mike. "Bar Chart Race, Explained." *Observable*, 19 Mar. 2021, observablehq.com/@d3/bar-chart-race-explained.

[16] KARATAŞ, ÖMER. "Make Your Own Custom Bar Chart Race with d3.js." *Medium*, Medium, 7 Mar. 2020, medium.com/@tarsusi/make-your-own-custom-bar-chart-race-with-d3-js-b7d6cfc4d0bd.

CONTRIBUTION:

Aatir Fayyaz: 33.33%

Kunlun Wang: 33.33%

Muhammad Saghir: 33.33%