Predicting Garbage Collector Invocation

Team Hawkeye

Methodology Followed:
Data pre-processing, Understanding the dataset, Feature Engineering, Building a suitable model and Applying to the cleaned dataset.

Data Pre-Processing:
Used Labelencoder from sklearn python library to encode the string data into numerical data. Ex: **gcRun**, **query token**.

Understanding the dataset:
We created some univariate graphs with matplotlib.pyplot and seaborn library of python to understand the nature of data.

Feature Engineering:

1. **CUMSUM_cputime** : Cumulative sum of cpuTimeTaken used as a feature because this feature is working as time series for gcRun.
2. **Freq_y** : For each token, mean of subtraction between initialUsedMemory and finalUsedMemory multipled by cpuTimeTaken used as frequency of token in the corpus.
3. **Diff_y** : For each token, mean of subtraction between initialUsedMemory and finalUsedMemory used as average memory taken by each token query.

Features Selected:

1. initialUsedMemory
2. initialFreeMemory
3. query token
4. gcRun
5. cpuTimeTaken
6. freq_y
7. diff_y
8. CUMSUM_cputime

Model Description:

LSTM(long-short term memory) : When you are dealing with sequential data or data with temporal relationship. This can be through time, so any time series data. The most popular application right now is actually in natural language processing, which involves sequential data such as words, sentences, sound spectrogram, etc. So applications with translation, sentiment analysis, text generation, etc.

First we converted our data into sequential form so that we can put LSTM model on it and then we used it in training as follow :

1. LSTM : we used our first LSTM model on data with all selected features to get finalUsedMemory for each step. So that we can use it as initialUsedMemory for next step.
   The rmse error on validation data was **0.122.**

2. LSTM : we used our second LSTM model on data with one extra feature (initialUsedMemory) to get finalFreeMemory for each next step. So that we can use it as initialFreeMemory for next step.
   The rmse error on validation data was **0.120.**

3. Xgboost classifier : we used this classifier to get gcRun value (True or False). The classification was on the imbalanced classes so we used scale_pos_weight(a parameter of xgboost that is used for imbalance classes) to get some good results.
   The F1 score and accuracy on validation data was **76.81%** and **92.72%.**

We created two files one was better in RMSE score and the other was better in F1-beta score so we take the initialFreeMemory from first file and gcRun from the other file and able to get good score relative to our previous approaches.

There are 3 .ipynb code files first run the Better-F1-beta and then run the Better-RMSE and then run the Combined-submission file to generate Final submission.