GS Quantify 2017

Garbage Collector Invocation

Insights obtanied from data

- Whenever InitialFreeMemory goes down , GC invokes. Though there was no particular threshold value for it
- For different token, rate of memory usage per CPU time was centred along particular point and the variation was not much
- So, memory usage per unit time was taken as constant for a particular token.
- Whenever GC invokes, FreeMemory goes up and UsedMemory goes down but by varying extent.
- Since first row of test data was given, second row could only be obtained with help of first row and so on..
- For example in case when GC does not invoke, finalMemory of first row would be InitialMemory of second row.
- But Final memory was not given in test data , it had to be modelled using available information.
- Same would not be case when GC provokes , finalMemory of first row would not be same as InitialMemory of second row.
- It was required to model when GC provoked actually

Preparation of Auxilary Data

As stated earlier , memory usage per unit time was taken as constant for a particular token. So a new data was made which included 91 tokens and ratio of memory used and CPU time. To do so , first of all row where GC invoked were excluded. Difference was then calculated using difference between finalFreeMemory and initialFreeMemory. Invoed cases were excluded beacuse this difference would not give the same memeory used in their case. This difference was divided by CPU time. And data was saved in CSV form for future use.

Feature Engineering

Features were required to build a model that could predict if GC will invoke or not. Looking at almost empty test dataset, it was quite certain that feature engineering would be backbone of model. Auxilary data was merged with original training data which added a feature ‘mean_ratio’ to it. Considering our previous assumption, estimated finalUsedMemory and finalFreeMemory were obtained:

$$Est.\ finalUsedMemory = InitialUsedMemory + CPUtime * mean\_ratio$$
$$Est.\ finalFreeMemory = InitialFreeMemory - CPUtime * mean\_ratio$$

These estimated value were very close to original finalUsedMemory and finalFreeMemory. The only motive was calculating estimated value was that test data did not have original values. Further we require same set of features for both train and test data. So optimal choice was to estimate the values for both data.Total Energy which comprise of sum of these two energies was another feature.

Apart from that, two ratios : i) ratio of initialFreeMemory and TotalMemory and ii)ratio of estimatedFinalFreeMemory and TotalMemory ,were used to build the model.

Model

Ratio of FALSE to TRUE in case of gcRun was very high (more than 10). There were primary two requirements of model:

i) it doen't get biased toward FALSE and starts predicting only FALSE , which is very common in case of imbalanced data.
ii) we are not quite sure about the strength of model , all the features are provided manually and none of it was itself in data. So strong classfier was needed.

Extreme Gradient Boosting is one of the classifier that could satisfy both the requirements. It takes a number of weak learners and join them to form a strong one. Xgb has an inbuilt paramter scale_position_weight which is specifically for the imbalanaced binary classifcation. Setting scale_position_weight to high value forces model to care of minoriy class also. It seems Xgb is the perfect weapon one would have got to tackle such a situation.

Optimal paramteres for the model were obtained using simple grid search with intelligent initial guesses.

Again the auxilary data is merged to test data in order to get the same set of features as train data one by one.

Prediction

For the very first row, finalUsedMemory and finalEmptyMemory were estimated using the same same equation stated earlier. Total was calculated by summation of both and in same way tha two ratios were determined. For the next row, InitialUsedMemory was same as finalUsedMemory of previous row and initialEmptyMemory was same as finalEmptyMemory of previous row. Calculation of whole set of feature became simpler in this iterative fashion. But, this was not all , case was in different when it was predcited GC would invoke. Once all the feature were obtained for a single row, gcRun was predicted on basis of those feature. If the prediction was FALSE , the iteration contiune and feature for next rows are calculated using this row and above stated equations. But if TRUE is predicted, which means when GC invokes, then the above equations does not hold and values of Estimated finalUsedMemory and finalEmptyMemory were updated and again they were used for calculation in next row. This way the iteration goes on till last row of test data.

This was how a whole bunch of predictions were made in iterative fashion from test data containing only one complete row.