PaytmLabs Challenge

Basic Processing

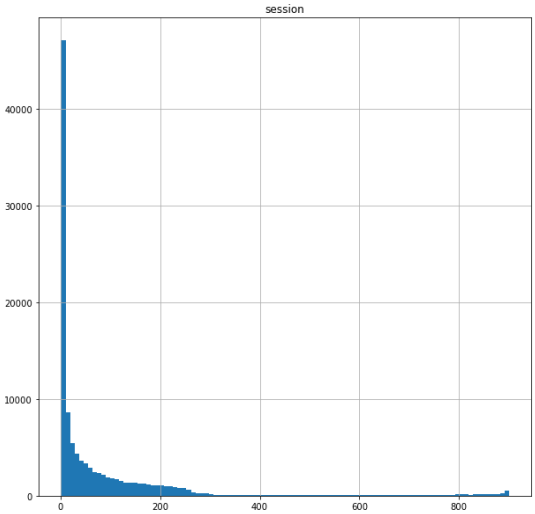
1. Session statistics

Take 15 minutes as a fixed session time window, here is the statistics: (session_analysis.py). If the session length is 0, I will treat it as 1, even though it is not a very well approximation.

```
       session(second)       url_count
count   113370.000   113370.000
mean    89.792273    8.193861
std    168.489399    54.157797
min    1.000000     1.000000
25%    1.000000     2.000000
50%    20.000000    3.000000
75%    103.000000   7.000000
max    900.000000   8016.000000
```

Average session length ≈ 90 seconds,
Average unique URL visits per session ≈ 8.2, check the detailed visits count for each session by "ip_session.data"
IP with the longest session times is '220.226.206.7', total session length is 5894 seconds
The single longest session time is 900 seconds

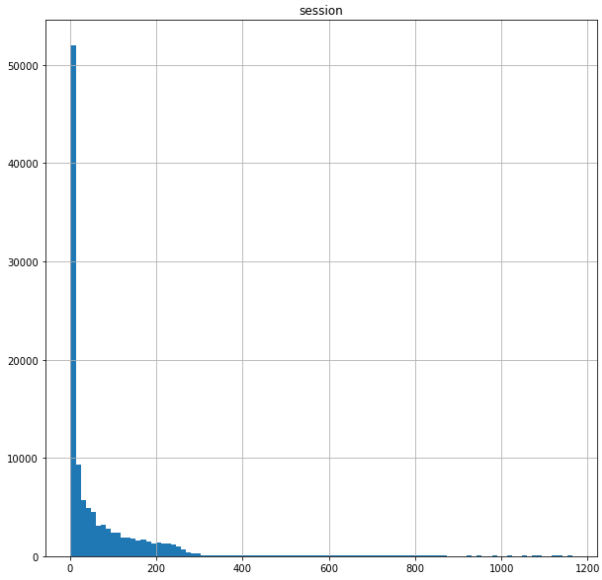
Here is the histogram graph for session length, it roughly follows the power law distribution or the Poisson distribution:



Another choice: If the time gap between two adjacent requests is more than ***10 minutes***, we can say a new session starts. Here is the statistic:

```
       session        url_count
count  114229.000  114229.000
mean  76.063346  8.120180
std  142.018872  56.225612
min  1.000000  1.000000
25%  1.000000  2.000000
50%  18.000000  3.000000
75%  95.000000  6.000000
max  1164.000  8016.000
```

Here is the histogram graph for session length, it also roughly follows the power law distribution or the Poisson distribution:



They are quite similar, result of the second method has smaller std.

Additional questions

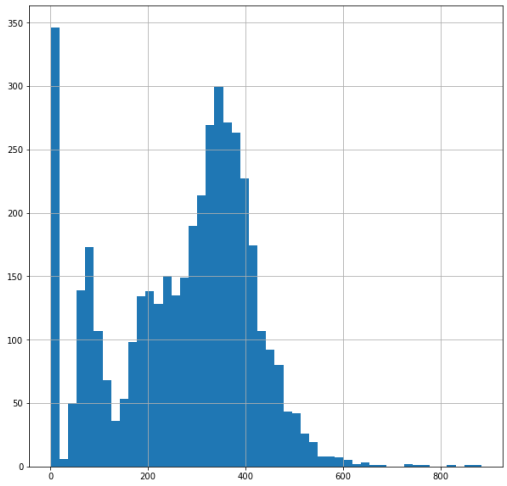
1. Predict the expected load in the next minute

1.1 Density estimation

If we can get the density estimation of the data, then we can sample values from the density as approximated predictions for future.
There are two kinds of methods for density estimation, parametric approach and non-parametric appoach. Generally, non-parametric approach named ***kernel density estimation*** has better performance, because it does not make any assumption about the data. I choose this method to fit the data.

If we choose one minute as the time unit, the dataset will be very small and sparse, so I use one second as time unit.

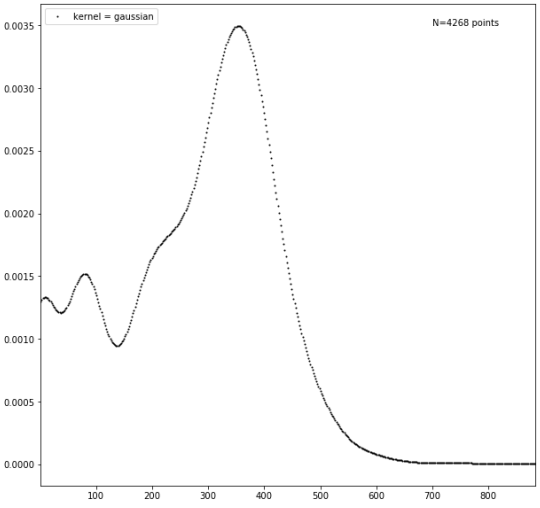
Here is the histogram of the request frequency per second:



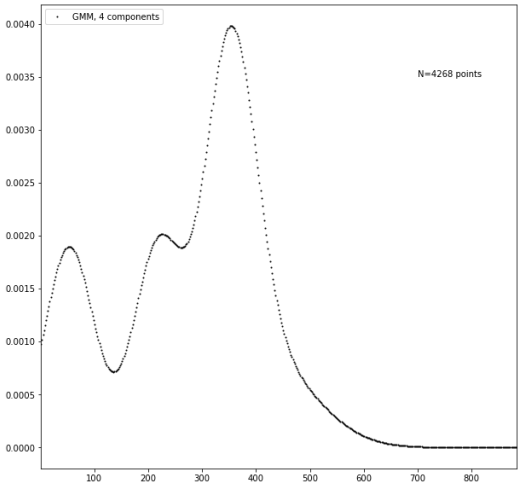
Here is the statistics of the request counts per second:

```
count     4268.000000
mean       271.414480
std        142.966698
min          1.000000
25%        175.000000
50%        304.000000
75%        374.000000
max        884.000000
```

Here is the density estimation of kde with Gaussian kernel, check predict_load\kde.py



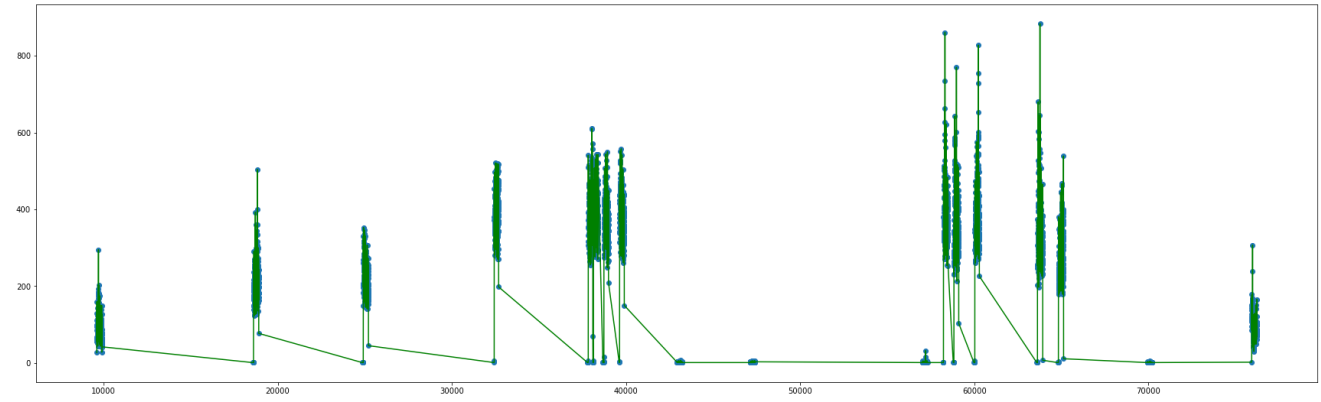
Here is the density estimation of the GMM

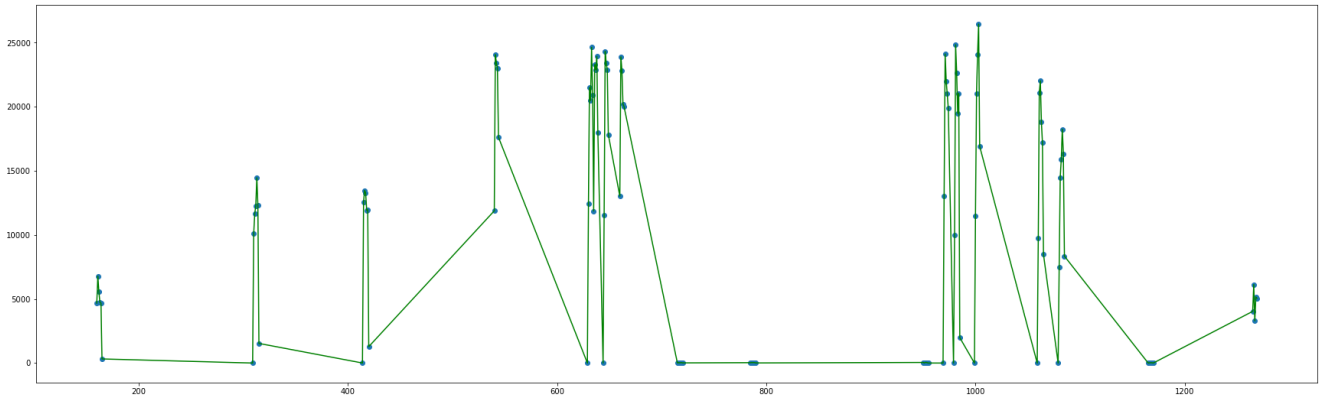


They are rather similar. We can do sampling from them to make predictions. See kde.py

1.2 Time series curve

Second as time unit and Minute as time unit





It is clear that it is not good choice to make sample based on these time series curve, there are too many gaps.

1.3 Regression and classfication

The source code is predict_load\cl_reg.py

1.3.1 Regresson

After analysing the all fields in a single request, it seems that only the time field has correlations with the users' behaviors. For example, it is obviously that there should be more requests in the afternoon than requests in early morning.

But due to the limited amounts of data, the correlation is not clear. Here is the correlation matrix (respect to count):

```
hour      0.070500
minute   -0.055574
second    0.004524
count     1.000000
```

It is clear tha only the original features are not enough, we need more features.

We can use kernel models to expand the feature space, such as SVR.
mean squared error of test dataset: 68.4

We also can employ models that can directly learn features tranformations from data, such as GBM, Randomforst, ANN
mean squared error of test dataset using GBM: 67.7
mean squared error of test dataset using Randomforest: 70.35

When making prediction, we use the average of the outputs of the three models.

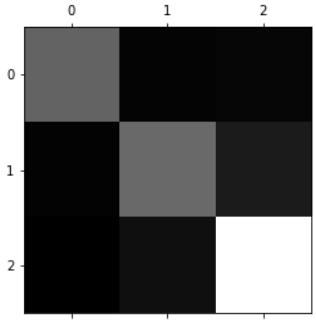
1.3.2 Classification

We can discretize the request counts into some buckets, and we assign a label for each bucket. We turn the regression to a classification problem. When making predictions,, we can retrieve a random number as predictions from the bucket which current time is assigned to.

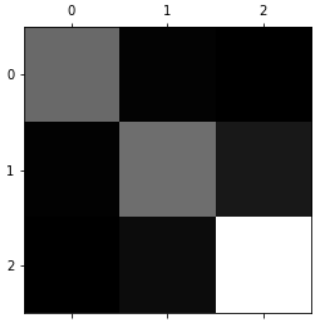
The difficulty is how to discretize the request count correctly and effectively: how to make sure the data in buckets are balanced? how to make sure the bias is acceptable?
The current performance is not reasonable, more work needed. Here is a brief summary.

Simply we can use three buckets [1, 150), [150, 300), [300, ~] labelled as 0, 1, 2. Three buckets are not likely to be sufficient, more work are needed to refine the bucket intervals.

If we use RandomForest classifier, the best test accuracy is 87.9%, here is the confusion matrix:



If we use SVM classifier, the best test accuracy is 88.4%, here is the confusion matrix:



Predictions can be made by randomly selecting a value from the buckets.

1.4 Sequential Learning

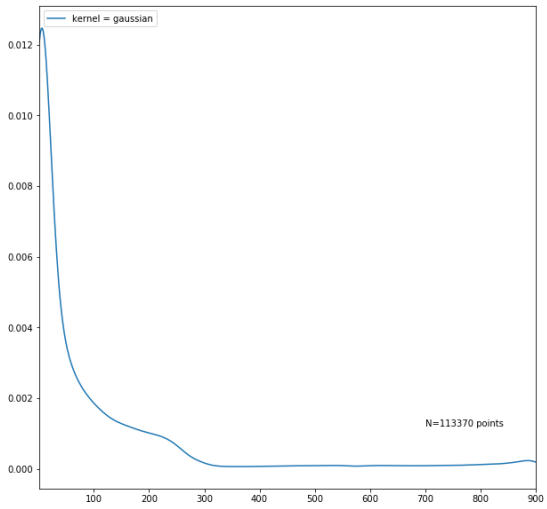
Sequential models might have better performance if there are more data, such RNN, LSTM, GRU. I have not implemented them becuase it is unlikely that they will perform better using these data.

2. Predict the session length for a given IP

Here is the statistics of the session legnth for IPs (predict_ip_info\ip_session_kde.py):

```
count   113370.000
mean        89.792273
std        168.489399
min          1.000000
25%          1.000000
50%         20.000000
75%        103.000000
max        900.000000
```

Use kernel density estimation estimate the probablity distribution of the session length, here is a showing graph.



We can sample values from this distribution as predictions.

3. Predict the number of unique URL visits by a given IP

Here is the statistics of the numer of unique URL visits by IPs (predict_ip_info\ip_url_kde.py):

```
count      90544.000000
mean          10.259520
std          159.893034
min            1.000000
25%            2.000000
50%            3.000000
75%            8.000000
max        32174.000000
```

Use kernel density estimation estimate probablity distribution of the unqiue URL visits by a given IP: