How to generate training data effectively

Jason Hua

6/5/19

Summary

- Progress on active learning (some positive result)
- Implemented the idea of computing distance for generating new sample (some negative result)

Why do we want to answer this question?

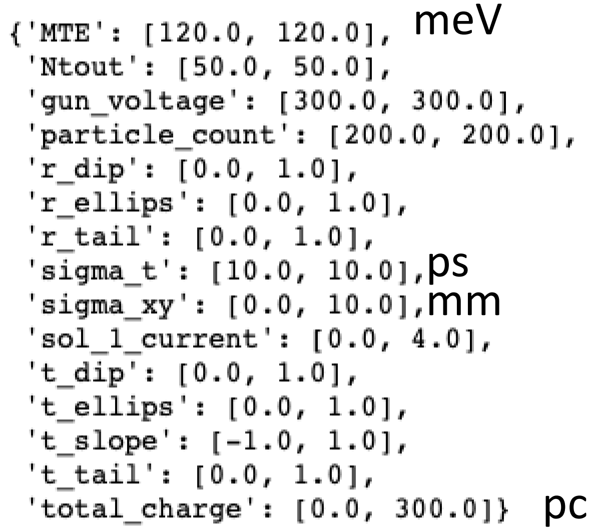
- We want to systematically compare how different ML architecture work. (Tree, GP, Neural Network)
- Auralee has pointed out that there is currently lacking a systematic study of what currently existing ML architecture suits accelerator physics.
- This systematic study can have a broad impact as people may often refer to this as a starting point.
- However, unless we can get efficient training data, we cannot do a such study on a reasonably large accelerator.

Application to GA

- We need to train the surrogate model used in GA
- We are essentially doing an optimization problem, and if we don’t want to miss an global minimum that is sharp. Then, we need to make sure the surrogate model is really sensitive.
- We cannot rely on random uniform sampling, as random sampling means the surrogate model maybe really sensitive around some region and maybe not so around some other region.
- In order to make sure the surrogate model can do prediction well across the entire physical range, we need to have better sampling technique.

Toy problem--dcgun

Input Parameter



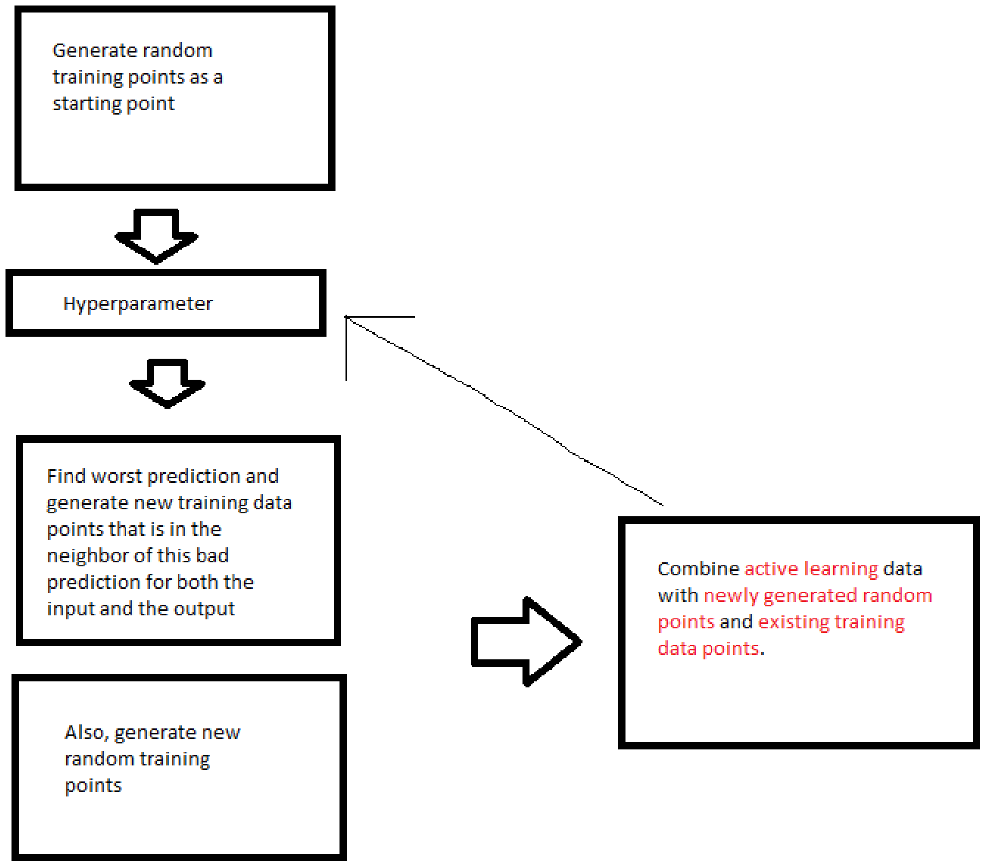
```
{'MTE': [120.0, 120.0], meV
 'Ntout': [50.0, 50.0],
 'gun_voltage': [300.0, 300.0],
 'particle_count': [200.0, 200.0],
 'r_dip': [0.0, 1.0],
 'r_ellips': [0.0, 1.0],
 'r_tail': [0.0, 1.0],
 'sigma_t': [10.0, 10.0],ps
 'sigma_xy': [0.0, 10.0],mm
 'sol_1_current': [0.0, 4.0],
 't_dip': [0.0, 1.0],
 't_ellips': [0.0, 1.0],
 't_slope': [-1.0, 1.0],
 't_tail': [0.0, 1.0],
 'total_charge': [0.0, 300.0]}  pc
```

Output

Emittance

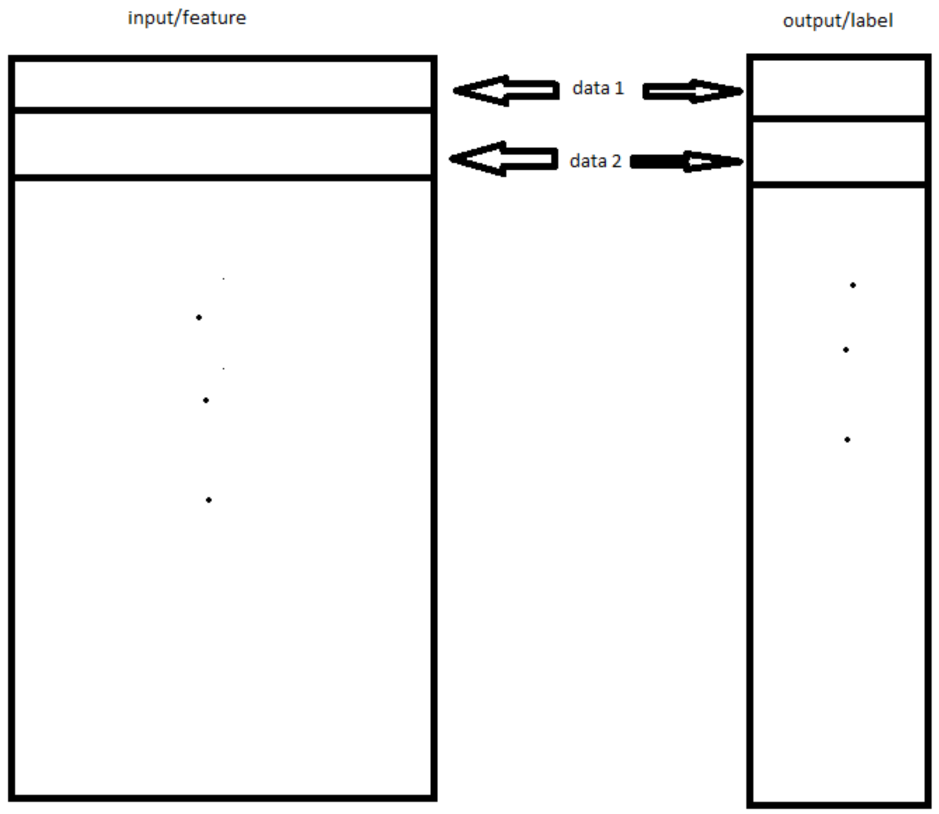
charge

Overall Scheme


Generate random training points as a starting point
Hyperparameter
Find worst prediction and generate new training data points that is in the neighbor of this bad prediction for both the input and the output
Also, generate new random training points
Combine active learning data with newly generated random points and existing training data points.

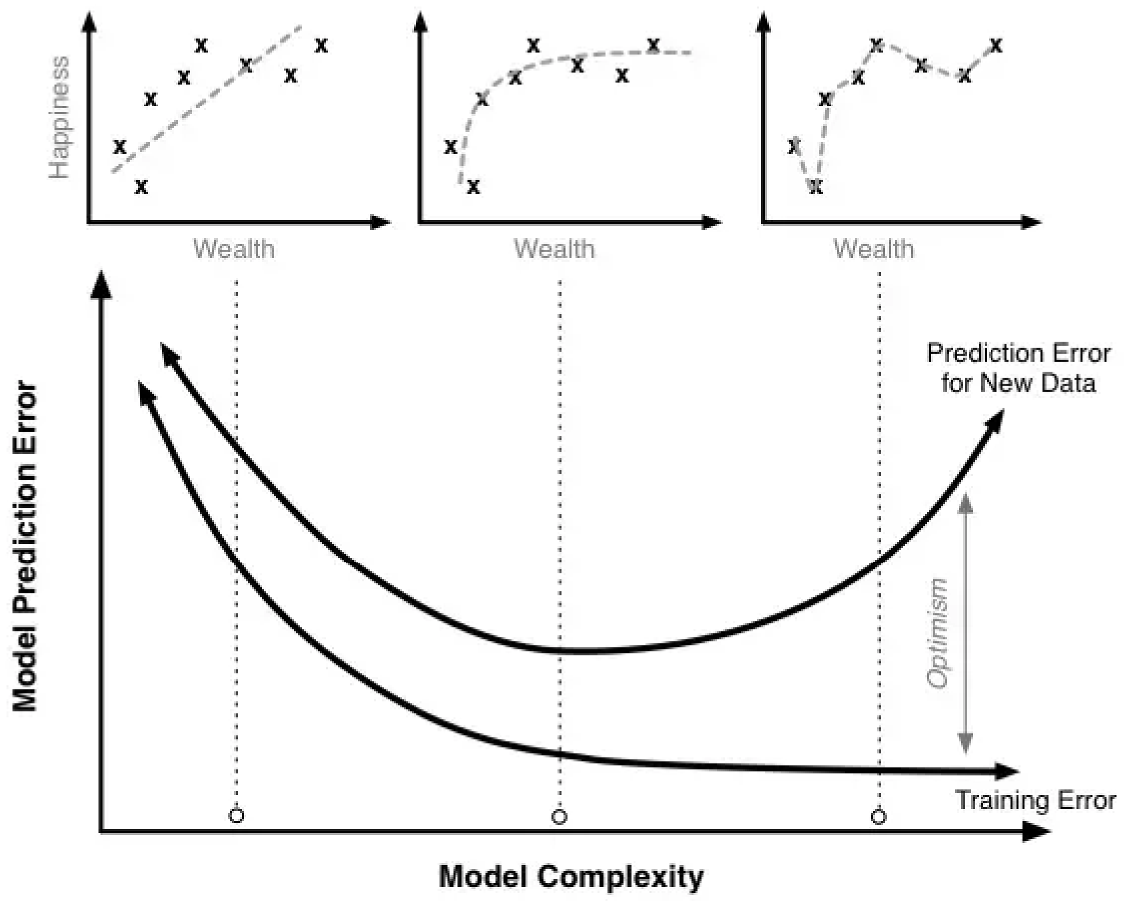
Active Learning

- Generate random training data
- The data looks like the following:
- The training input data looks like a matrix of size N*D_in, where N is the number of training data and D_in is the dimension of the input
- The training label looks like a matrix of size N*D_out, where N is the number of training data and D_out is the dimension of the output
- For this case, we begin with only 60 data points


input/feature
output/label
data 1
data 2

Then, we tune hyperparameters

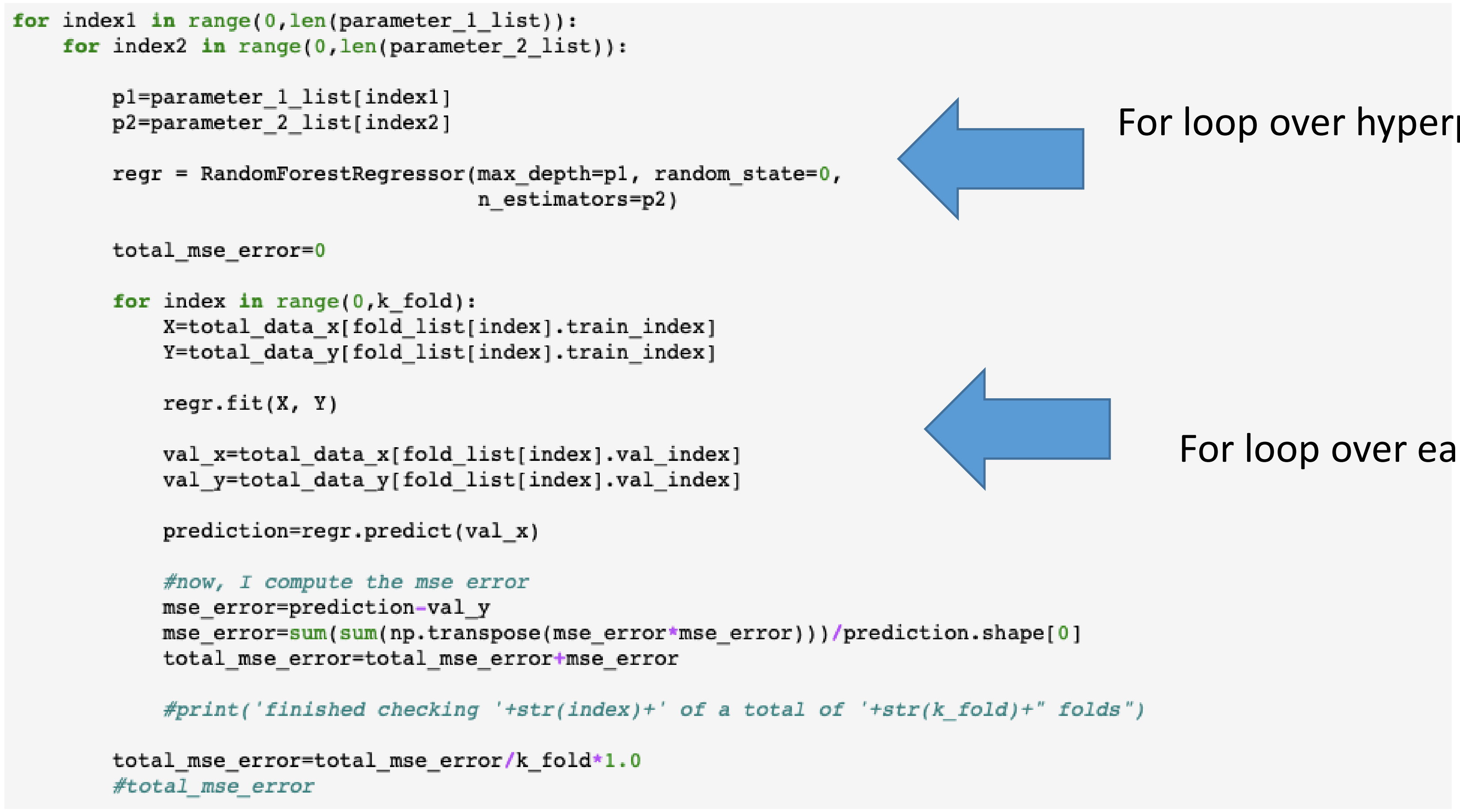
We use a random forest regressor for this case



Then, we tune hyperparameters

- We split the data into 3 folds
- There is no particular reason why we use 3 here, it just seems reasonable
- Each fold now has 20 data points

- For each round of tuning hyperparameters, we use two folds as training and one fold as testing, and the testing fold is changed from 1 to 2 to 3.
- We repeat this process for all possible hyperparameters.
- We find the hyperparameters with the lowest testing error.



```python
for index1 in range(0,len(parameter_1_list)):
    for index2 in range(0,len(parameter_2_list)):

        p1=parameter_1_list[index1]
        p2=parameter_2_list[index2]

        regr = RandomForestRegressor(max_depth=p1, random_state=0,
                                     n_estimators=p2)

        total_mse_error=0

        for index in range(0,k_fold):
            X=total_data_x[fold_list[index].train_index]
            Y=total_data_y[fold_list[index].train_index]

            regr.fit(X, Y)

            val_x=total_data_x[fold_list[index].val_index]
            val_y=total_data_y[fold_list[index].val_index]

            prediction=regr.predict(val_x)

            #now, I compute the mse error
            mse_error=prediction-val_y
            mse_error=sum(sum(np.transpose(mse_error*mse_error)))/prediction.shape[0]
            total_mse_error=total_mse_error+mse_error

            #print('finished checking '+str(index)+' of a total of '+str(k_fold)+" folds")

        total_mse_error=total_mse_error/k_fold*1.0
        #total_mse_error
```

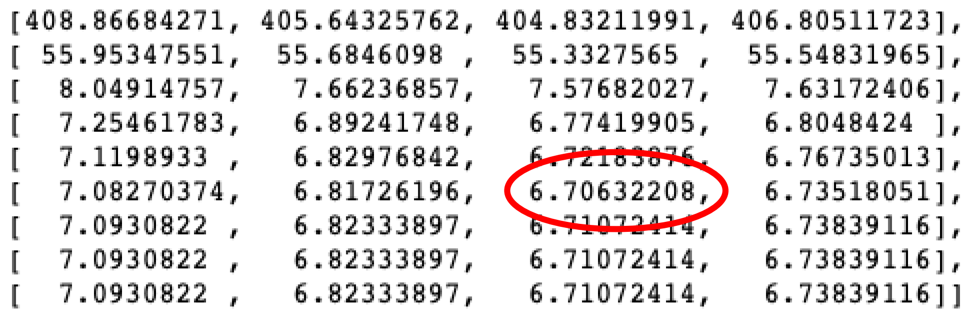
For loop over hyperparameter

For loop over each fold

Depth=10, size =400

Forest size

Tree Depth

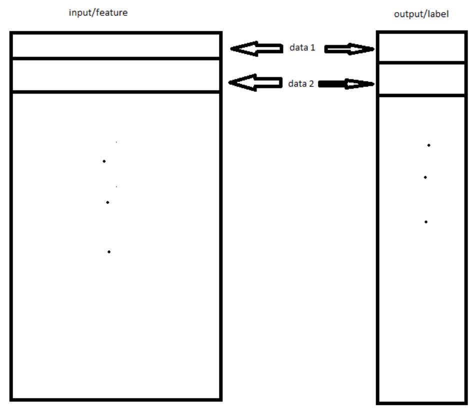


```
[408.86684271, 405.64325762, 404.83211991, 406.80511723],
[ 55.95347551,  55.6846098 ,  55.3327565 ,  55.54831965],
[  8.04914757,   7.66236857,   7.57682027,   7.63172406],
[  7.25461783,   6.89241748,   6.77419905,   6.8048424 ],
[  7.1198933 ,   6.82976842,   6.72183876,   6.76735013],
[  7.08270374,   6.81726196,   6.70632208,   6.73518051],
[  7.0930822 ,   6.82333897,   6.71072414,   6.73839116],
[  7.0930822 ,   6.82333897,   6.71072414,   6.73839116],
[  7.0930822 ,   6.82333897,   6.71072414,   6.73839116]]
```

Interestingly, the hyper parameter for tree depth is always around 10.
Maybe this is due to there are 10 input parameters.

Now, find the worst prediction

- We build the model using the hyperparameter we just learned.
- We have two folds as training data and one fold as testing data. Again, the testing data is shifted from fold 1 to 2 to 3.
- We find the worst prediction
- Because we always have this one-to-one correspondence, if we know the label, we know the input.

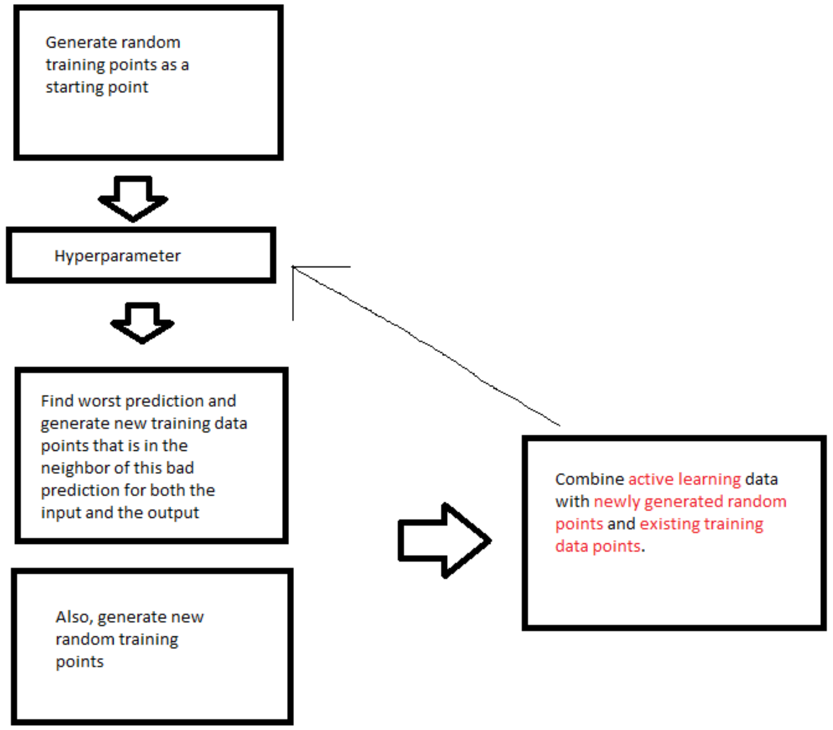


Generate new training data

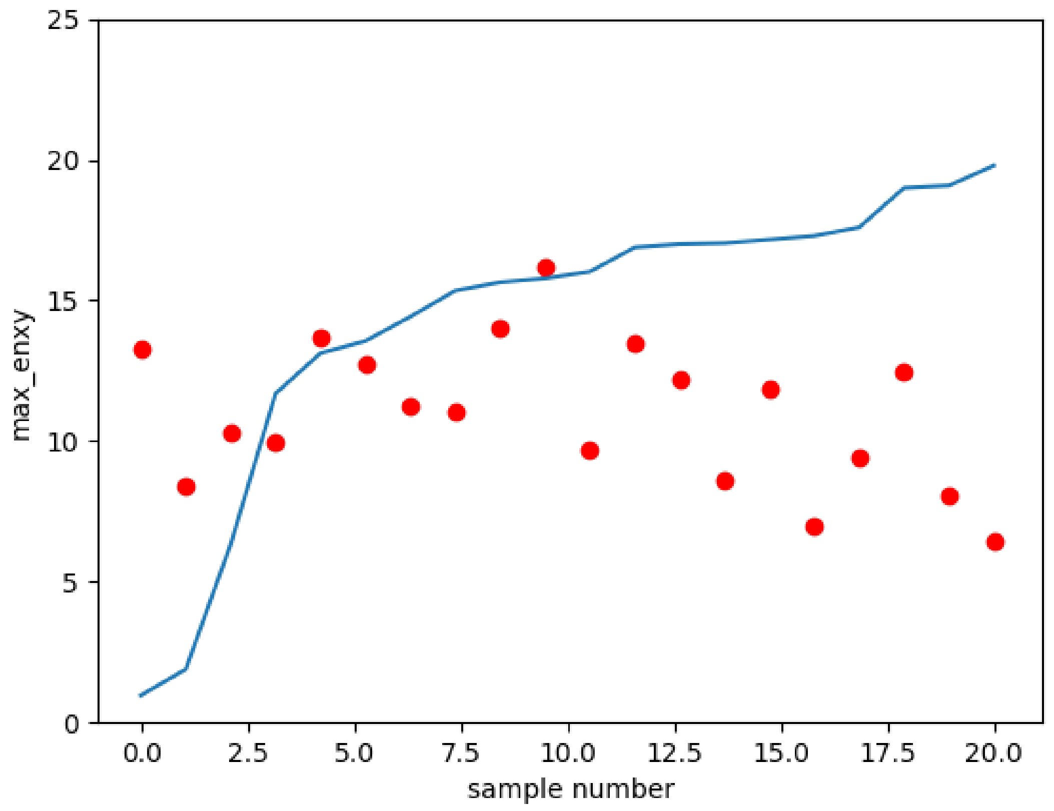
- We record which input gives rise to the bad output.
- We ask GPT now to generate a lot of input data near those input data we just identified.
- Once GPT is finished, if the output also lies in the bad ouput region, we record this new data point. And complete one active learning point.

- To make sure, we are also exploring out regions of data and not just fixing existing problems, we also ask the program to randomly generate new points.

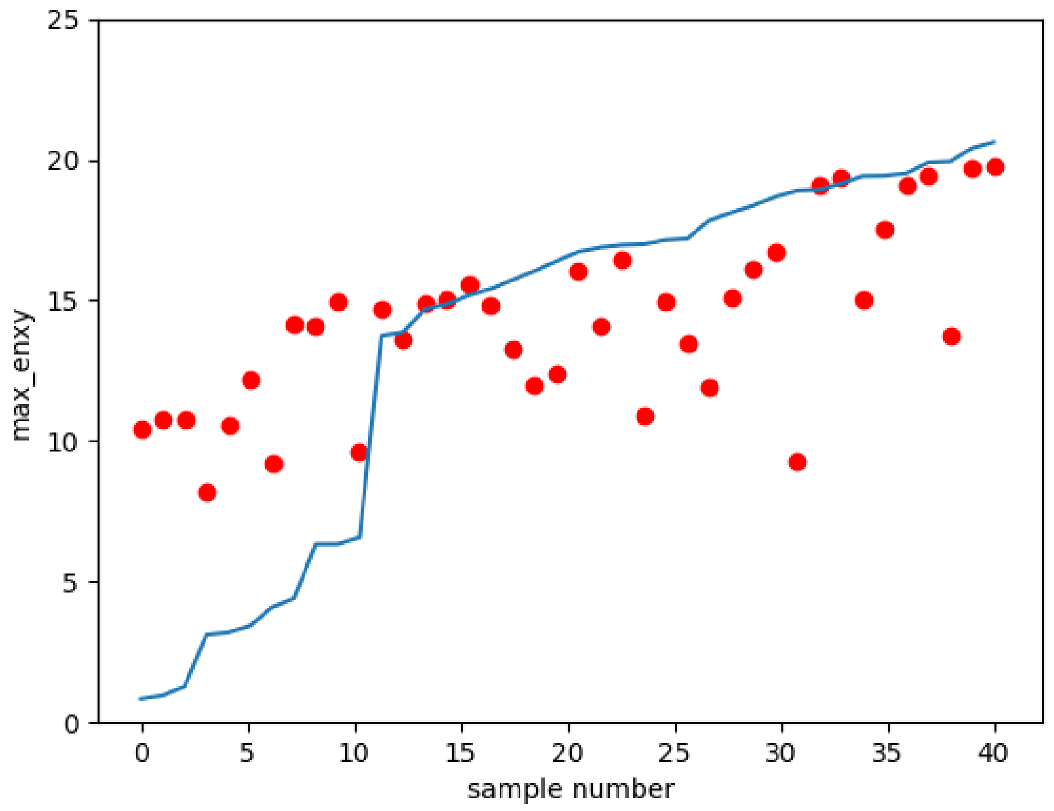
Overall Scheme


Generate random training points as a starting point
Hyperparameter
Find worst prediction and generate new training data points that is in the neighbor of this bad prediction for both the input and the output
Also, generate new random training points
Combine active learning data with newly generated random points and existing training data points.

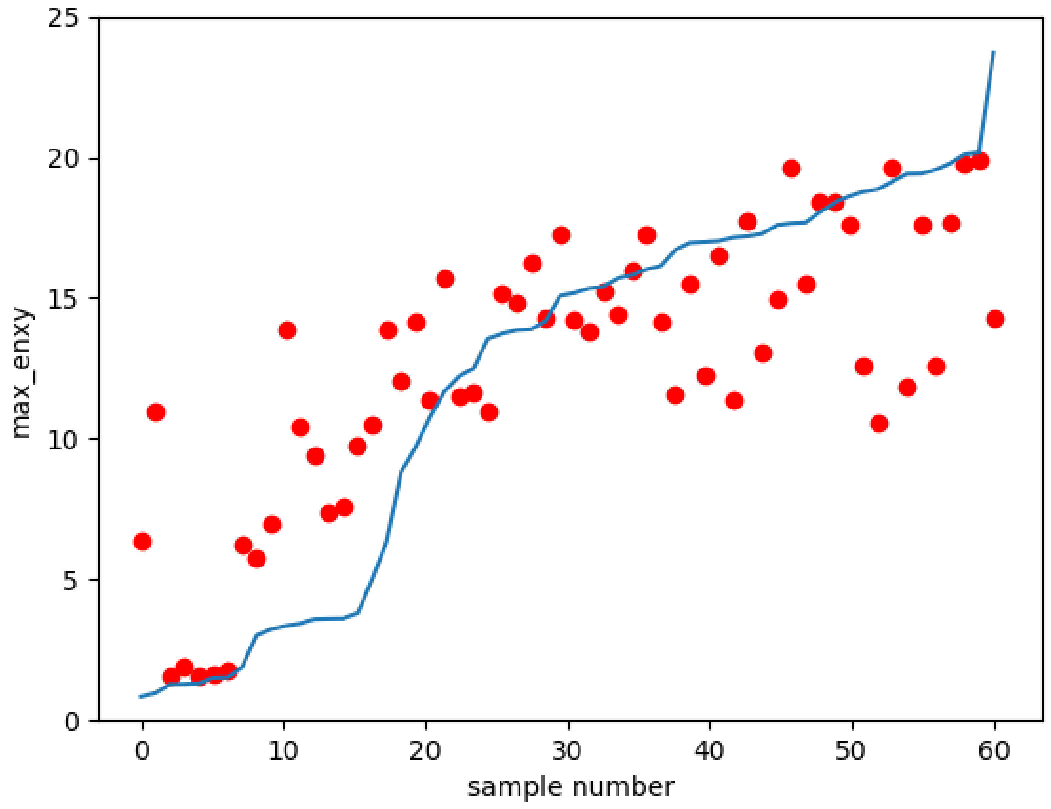
Result (using first fold as testing data)-iteration 0



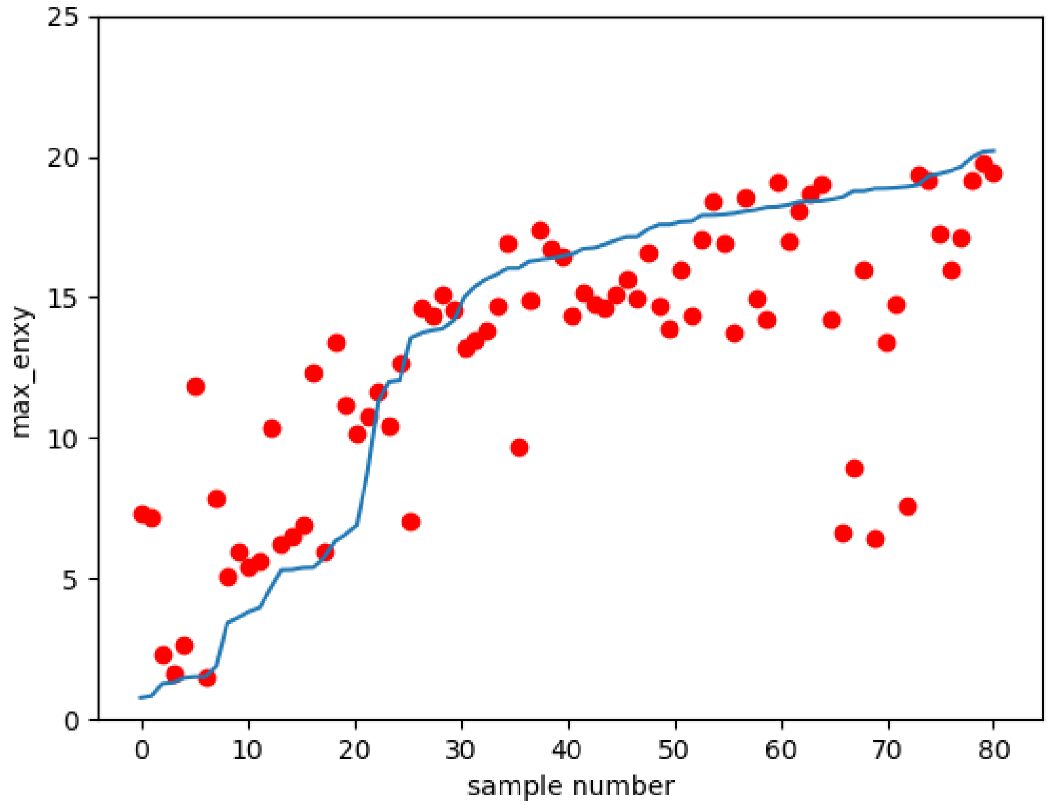
iteration 1



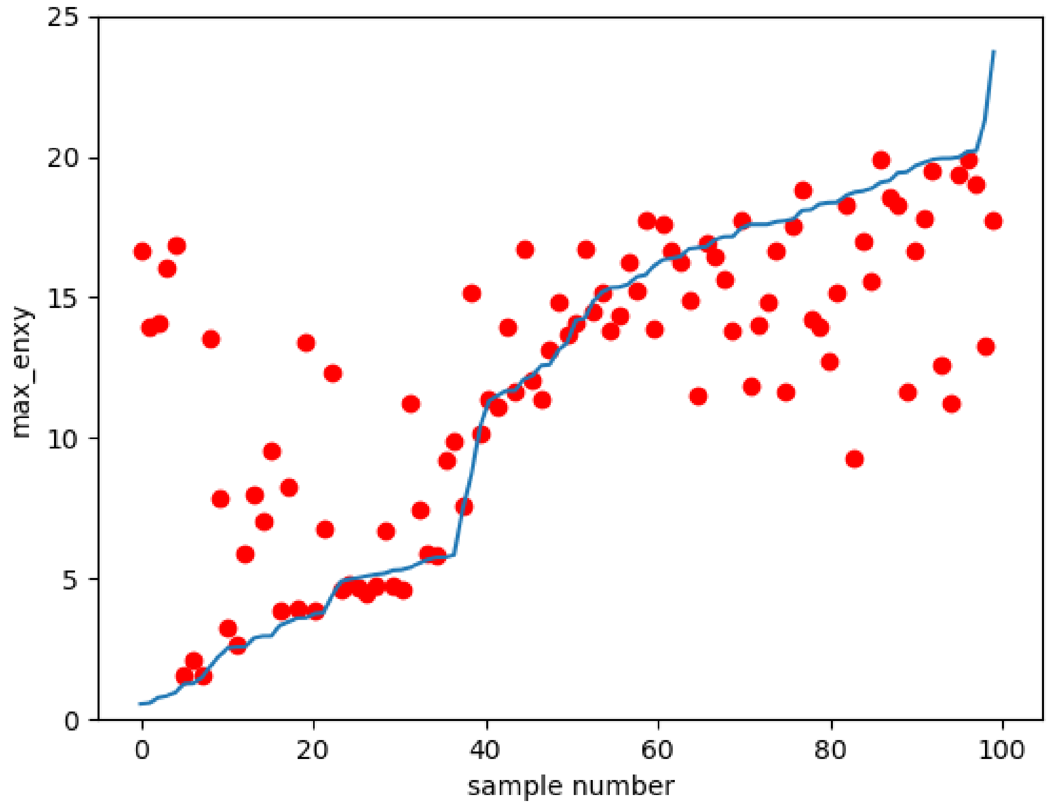
iteration 2



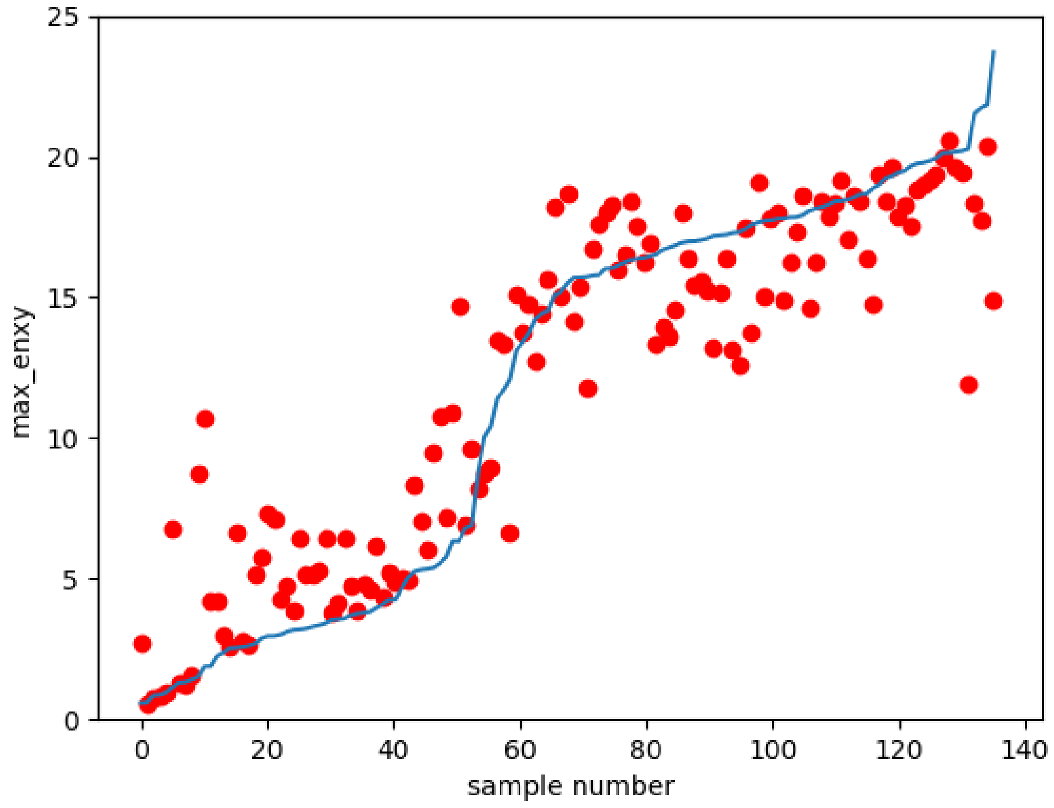
iteration 3



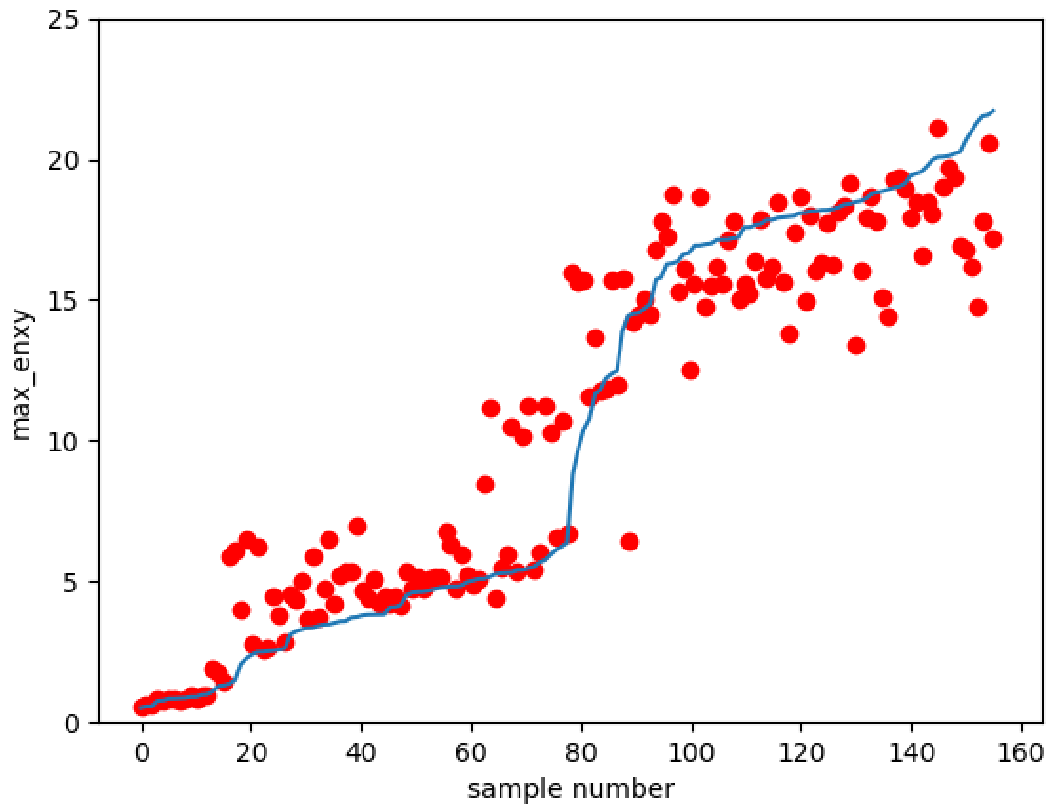
iteration 4



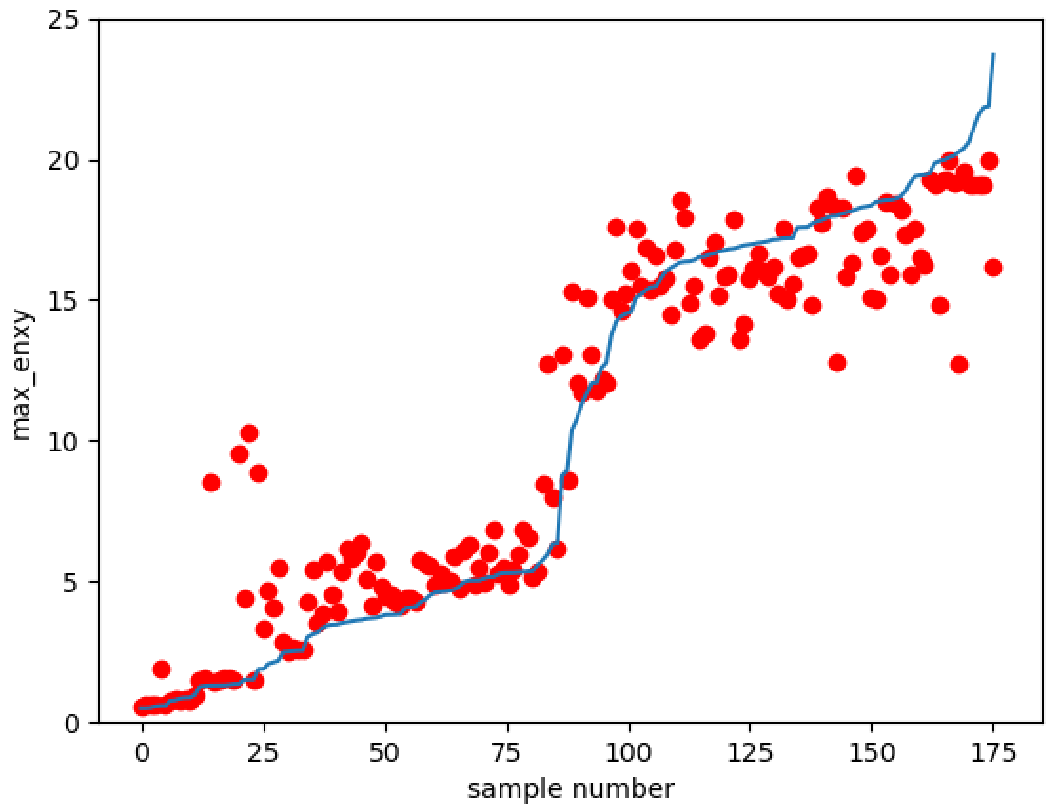
iteration 5



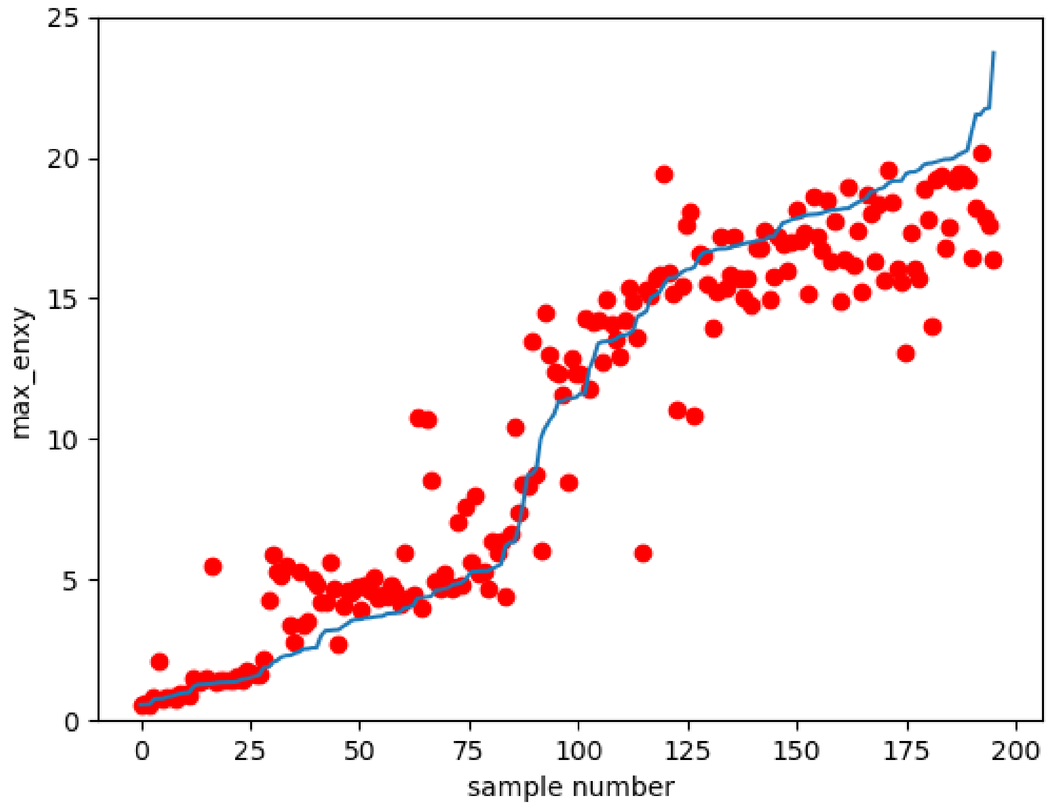
iteration 6


max_enxy
sample number
0
5
10
15
20
25
0
20
40
60
80
100
120
140
160

iteration 7



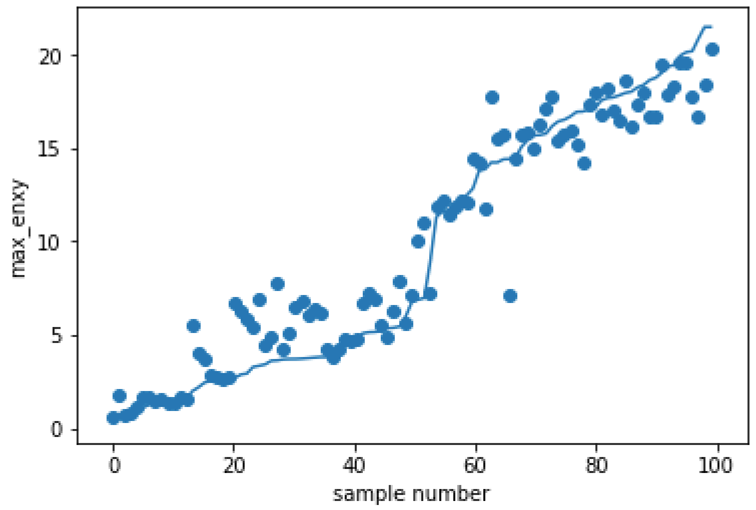
iteration 8



max_enxy
sample number
0
5
10
15
20
25
0
25
50
75
100
125
150
175
200

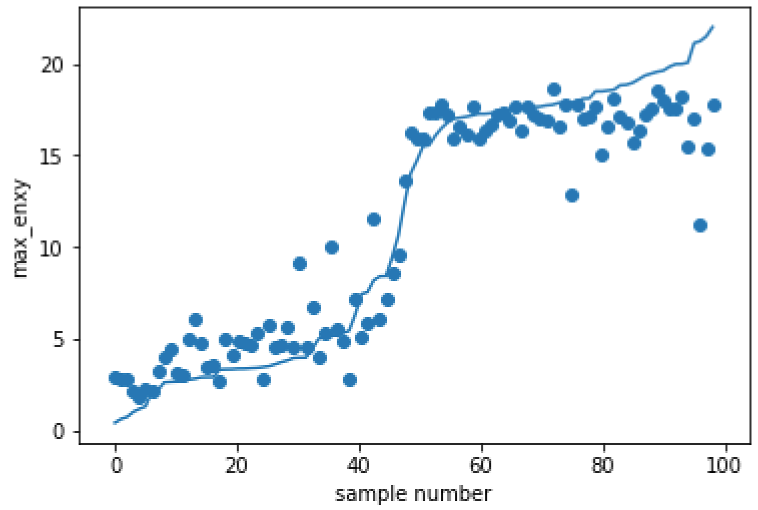
We truncated the active learn data from 600 to 500 to compare with a randomly generated data set of 500 points

Active Learning



Mse: 12.4620

Uniform Random Sampling



Mse: 13.906

Problem with this approach

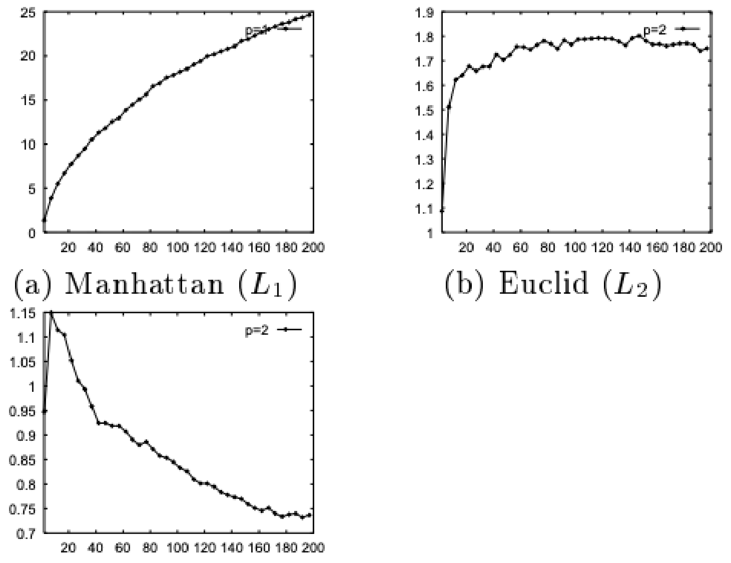
- This process can be slower.
- This is because when we generate the specific data points for active learning, we currently sample from a uniform distribution around the bad input and we need to ensure the output data is also what we want. This means sometimes we can have a GPT simulation whose result is not desirable.
- We can solve this problem by learning the input distribution around the bad input parameter.

Idea of generating new sample by computing distance.

- How do we measure the dissimilarity of two input vector in high dimension?
- Usually, we do so by the L2 norm.
- But this doesn’t work so well in high dimension.

What is the nearest neighbor in high dimensional spaces?

(Alexander Hinneburg, Charu Aggarwat, Daniel Keim)



(a) Manhattan (L_1)

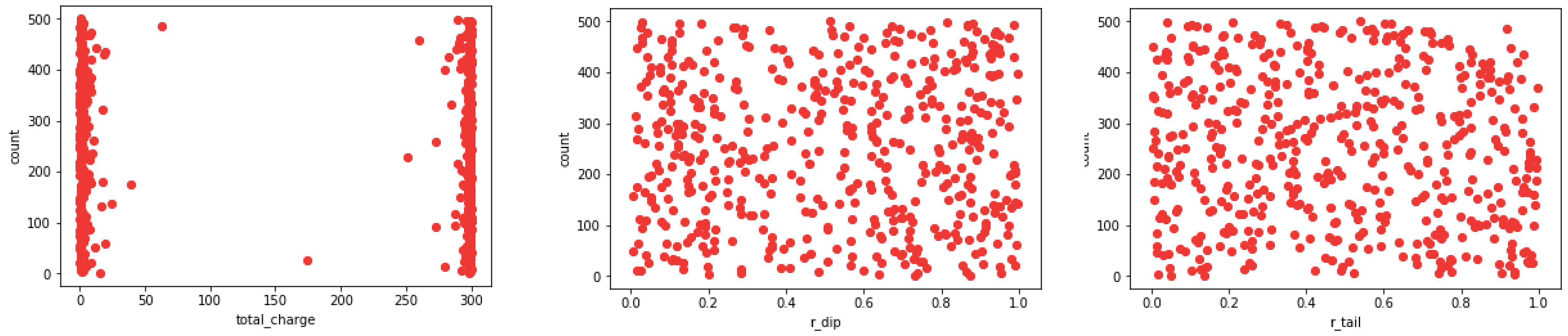
(b) Euclid (L_2)

(c) $L_3 Metric$

Figure 1: $|Dmax - Dmin|$ depending on d for different L_k metrics (uniform data)

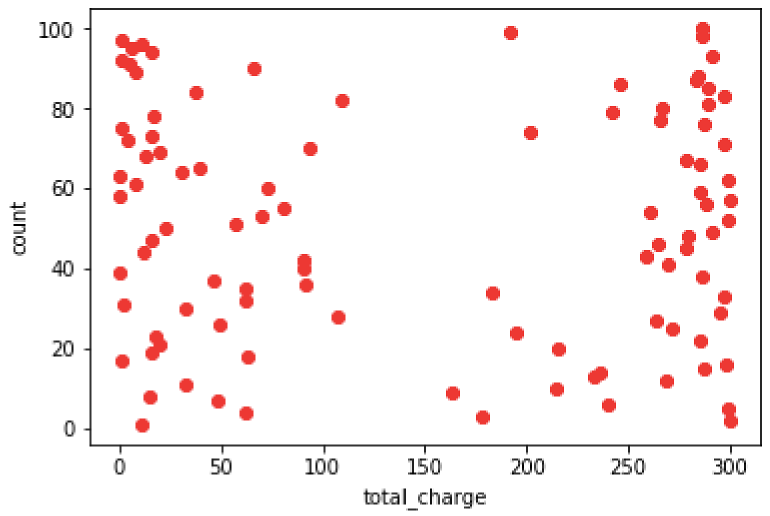
So, we want to use L1

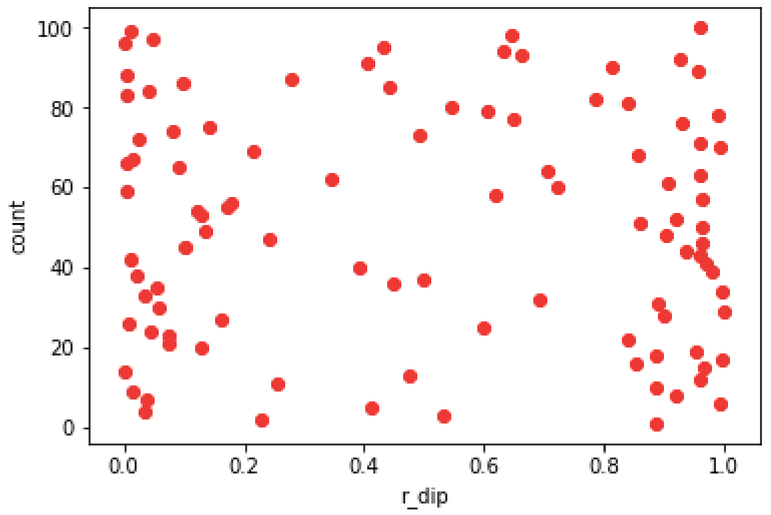
- Also, we need to scale the data, because currently the input data don’t have the same range. If we don’t scale the data, we get the input to be something like this (500 points):

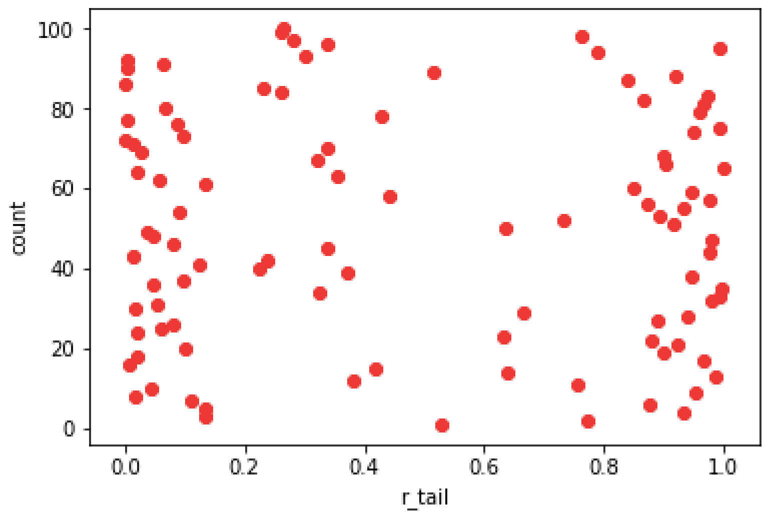


Input parameter with the largest numerical value gets pushed to the two extremes

Now, we scale all the input to be between 0 and 1 (100 points)

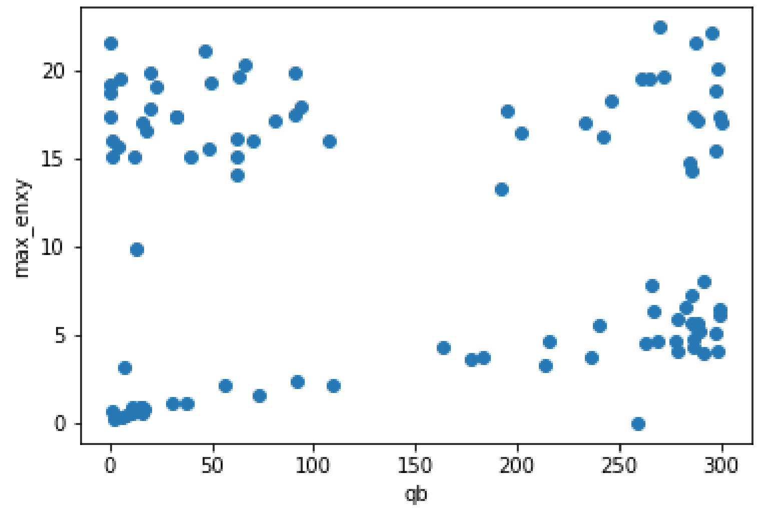






This helps a little bit but not enough,
You will see why on the next page

We need to be smarter



The output data all lives on the edge

The data lives on a high dimensional space, (there are 10 input parameters), this means all the volume is near the edge. If we just use distance as a measure, I don’t think we can solve this fundamental issue.

- I have uploaded all the program and data on box.
- If anybody wants to use it, email me, as the implementation is not super user friendly yet.