Assignment 4: Performance Metrics and Optimisation

Part 1: Performance Metrics in Regression

Exploratory Data Analysis (preprocessing)

In the diamonds dataset, the categorical type attributes are: cut, colour and clarity. To implement a regression algorithms to predict the price of round cut diamonds, it should convert all the categorical type attributes to numeric. Because regression algorithms can't not identify they grades, when they grades represent as a strings. I have done a bit research about cut, colour and clarity of diamonds. The result shows as follow: (from highest to lowest grades)

Round

	Premium Ideal	Ideal	Excellent	Very Good	Good	Fair	Poor
Total Depth %	59.5 - 61.9	59.1 - 62.3	58.5 - 63	57.5 - 63.5	57 - 64	56 - 67	<56 >67

The follow figure shows use python pandas to replace the categorical type attributes to numeric type attribute.

```
# replace cut column attribute 'Fair','Good','Very Good','Ideal','Premium' to 1 2 3 4 5
data=data.replace(['Fair','Good','Very Good','Ideal','Premium'],[1,2,3,4,5]);
# replace colour column attribute 'D','E','F','G','H','I','J' to 7,6,5,4,3,2,1
data=data.replace(['D','E','F','G','H','I','J'],[7,6,5,4,3,2,1]);
# replace clarity column attribute 'Fair','Good','Very Good','Ideal','Premium' to 1 2 3 4 5
data=data.replace(['I1','SI2','SI1','VS2','VS1','VVS2','VVS1','IF'],[1,2,3,4,5,6,7,8]);
```

To standardise all the inputs, because the reaction on the data of those attributes are particularly large. Those attributes have different weights, for example without standardise the attribute ‘depth’ will play a dominant role, another attributes likes ‘x’, ’y’, ‘z’ will just have tiny reflect on this data. So, we want to male those attributes have balance weights, to make every attributes does the same contribution on the data, and that is why we have to standardise all the inputs.

```
# Standardize the inputs

train_mean = train_data.mean()
train_std = train_data.std()
train_data = (train_data - train_mean) / train_std
test_data = (test_data - train_mean) / train_std

print(train_data)
```

Tune parameters

	Tuned parameter	Resons
Liner regression	set as default	
K-neighors regression	set k = 11	In theory, the larger dataset and k value we have, the higher accuracy we can get. I think set k = 11 is a suitable value for this data set.
Ridge regression	set as default	
Decision Tree regression	set max_depth = 9	The maximum depth of the tree is controlled by the max_depth parameter. If set the max_depth parameter, it will cause overfilling problem. If set the max_depth parameter too low it will be under-training.
Random Forest regression	set max_depth=9, random_state=0, n_estimators=500	For max_depth, is the same reason above. The aim for set the estimators to 500 is for increases number of trees in the forest. To improve the predictive accuracy.
Gradient Boosting regression	n_estimators= 500, max_depth= 9,	The aim for set the estimators to 500 is for increases number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.The maximum depth limits the number of nodes in the tree, to avoid overfilling problem.
SGD regression	set as default	
SVR	Set as default	
Linear SVR	set as default	
Multi-layer perceptron regression	set learning_rate_init=0.2	The default was 0.001, I did increase the learning_rate to 0.2 ,that is the step-size in updating the weights. It helps algorithm train faster. Otherwise when the maximum iterations (200) reached and the optimisation hasn't converged yet.

Results

	Execute on Time	R2	MSE	RMSE	MAE
Liner regression	0.214	0.91	1381602.89	1175.42	789.29
K-neighors regression	0.392	0.96	707704.67	841.25	446.10
Ridge regression	0.111	0.91	1381642.69	1175.48	789.62
Decision Tree regression	0.099	0.97	428606.90	654.60	355.12
Random Forest regression	36.131	0.98	341582.90	584.45	313.53
Gradient Boosting regression	85.885	0.98	288526.55	537.15	266.48
SGD regression	0.335	0.90	1548139.19	1244.24	821.88
SVR	68.140	0.50	8164863.24	2875.24	1356.06
Linear SVR	0.82	0.82	2901564.78	1703.40	995.8
Multi-layer perceptron regression	4.633	0.97	409409.55	639.85	365.88

MSE (mean_square_error)

Mean Square Error is refers to the square of expect value of the difference between the predicted value and the real value. MSE can evaluate the change extent of the data, the small MSE values it have, that means the better accuracy of the prediction model it have. Compare all the regression algorithms above table, it shows Gradient Boosting regression have the lowest MSE.

RMSE (root mean square error)

In the above MSE, we squared the expected value. In this way might lead to dimensional problems, such as if the difference of diamond prices is $1000, after square it will become $1000000. Is better to square root MSE to shows the deviation between the predicted value and the value in a direct way. The Gradient Boosting regression has the lowest MSE will also have the smallest RMSE.

MAE (mean absolute error)

Mean absolute error is the average of the absolute error. Which can better reflect the actual situation of the prediction error Once again, the Gradient Boosting regression have better performance.

R-Squared

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

(Residual Sum of Squares)

(Total Sum of Squares)

$$R^2 = 1 - \frac{\sum_i (\hat{y}^{(i)} - y^{(i)})^2}{\sum_i (\bar{y} - y^{(i)})^2}$$

R-Squared is another way to measure the performance of the regression algorithms, from the above figure it shows the numerator is the predicted error of our model and the denominator is the baseline model error. Which means R^2 <= 1, if R^2=1, it means the prediction model will not have any

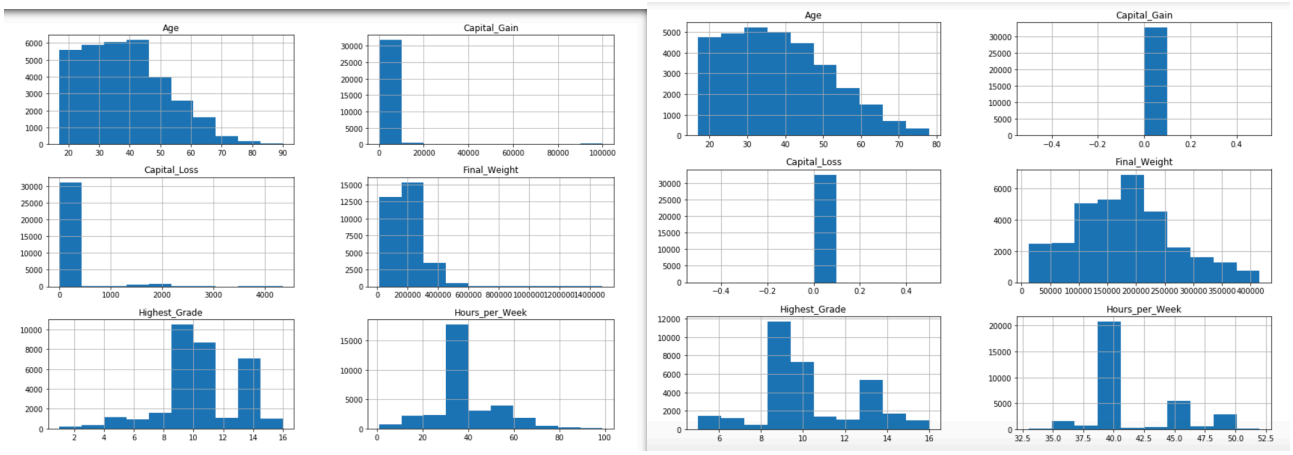
mistake, the larger r^2 value it have the better performance of the models it does. When the R^2 = 0, it means the prediction model is the same as the baseline model. If the R^2 < 0, that means the prediction model is worst than the baseline model. Or maybe there is not any liner regression in this dataset. In the table above, it shows Random Forest regression and Random Forest regression are have the largest R^2 values.

In conclusion, it shows there have a strong liner regression in this dataset. Gradient Boosting regression and Random Forest regression have the best performance overall. They are a type of decision tree regression, a random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The Gradient Boosting regression is allows for the optimisation of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. If those two regression increase their n_estimators parameters, it might have better performances. But if consider the time factor, Decision Tree regression is better option. It preforms extremely fast training speed, and achieve high performance.

Part 2: Performance Metrics in Classification

Exploratory data analysis (preprocessing)

Outliers



The top right figure shows the distributions of numeric attributes in adults dataset. On the final_Weight figure and hours_per_Week figure it shows the outliers are exist, the outlier values are too far away form the group. There are a math function to judge which value is the outlier. That is the values those who smaller than lower quantile - 1.5*IQR or larger than upper quantile + 1.5*IQR. After I find out all the outliers, I did set the outliers back to the bench mark value (lower quantile - 1.5*IQR, upper quantile + 1.5*IQR). And the result shows on the top left figure. One of interesting thing I find out is Capital_Loss and Capital_Gain attribute are to not suitable for dealing with outlier, because after they dealing with outlier the values in the attributes are all 0, that means the attribute is become useless, will not have any impact for classification. Finally I design to leave the Capital_Loss and Capital_Gain attribute as original.

Missing values

One simple line of code: “data.isnull().values.any()” can easily check is there are any missing values in the dataset. But before doing this I have to change all the ‘?’ Values to np.nan, otherwise python cannot identify ‘?’ as a missing value. After I find out the whole dataset have about 5% missing datas, it will not have much impart for the final classification, but I design to replace all the missing values with mode. I think mode is more suitable than mean, for consider about the capital gain and capital loss attributes use mean will set the

missing values around 10000. But there are over 90% of datas are don't have any Capital_Gain and Capital_Loss. To replace with mode values might be the best to return to the real situation.

Convert numeric to categorial (Discrete)

```
#convert numeric to categorical

data["Age"]=convertCategoricalToNumeric(data["Age"],8)
data["Final_Weight"]=convertCategoricalToNumeric(data["Final_Weight"],10)
data["Highest_Grade"]=convertCategoricalToNumeric(data["Highest_Grade"],5)
data["Hours_per_Week"]=convertCategoricalToNumeric(data["Hours_per_Week"],5)
```

```
def convertCategoricalToNumeric(data,binNum):

    data2=data.copy()
    datamax = int(data.max())
    datamin = int(data.min())
    print(datamax,datamin)
    binRange=int((datamax-datamin)/binNum)

    labels = ["{0} - {1}".format(i, i + binRange-1) for i in range(datamin, datamax, binRange)]
    data2 = pd.cut(data, range(datamin, datamax+binRange, binRange), right=False, labels=labels)

    return data2
```

I choose attribute Age, Final_weight, highest_grade and hours_per_week those four attributes for doing convention. If check out the data uniques of those attributes we can see there are lots of different type of values in those attribute. When we doing classification, we cannot just covert every single values to a categorial, because there are too many, it leads to produce many redundant values and those categorial type values are useless. To fix this problem, I think is best to split the numeric datas into several different categorial type. After run through the above code I wrote, the possible values in age attribute shows as follow [45 - 51, 38 - 44, 52 - 58, 24 - 30, 31 - 37, 17 - 23, 59 - 65, 66 - 72, 73 - 79]. By the way, for different attributes will split into different number of categorial type, it only depends on the range of the values. For example the range of hours_per_week is 19, then split into 5 different category should be enough. If spilt more, the bin size will be reduce, and is easy to produce redundant values.

Performance Table

	confusion matrix	accuracy	Precision	Recall	F1-score	AUC
KNN	[2257, 244 311, 464]	0.84	0.88 0.67	0.91 0.60	0.89 0.63	0.75
Naive Bayes	[2285,196 430,345]	0.81	0.84 0.64	0.92 0.45	0.88 0,52	0.68
SVM	[2358, 123 371 404]	0.85	0.86 0.77	0.95 0.52	0.91 0.62	0.74
Decision Tree	[2170, 311 324, 451]	0.80	0.87 0.59	0.87 0.58	0.87 0.59	0.73
Random Forest	[2372 109 358 417]	0.86	0.87 0.79	0.96 0.54	0.91 0.64	0.75
AdaBoost	[2280 201 295 480]	0.85	0.89 0.70	0.92 0.62	0.90 0.66	0.77
Gradient Boosting	[2357 124 337 438]	0.86	0.87 0.78	0.95 0.57	0.91 0.66	0.76

	confusion matrix	accuracy	Precision	Recall	F1-score	AUC
Linear discriminant analysis	[2358 123 490 285]	0.81	0.83 0.70	0.95 0.37	0.88 0.48	0.66
Multi-layer perceptron	[2327 154 353 422]	0.84	0.87 0.73	0.94 0.54	0.90 0.62	0.74
Logistic regression	[2338 143 463 312]	0.81	0.83 0.68	0.94 0.41	0.89 0.51	0.68

Is accuracy the best performance metric to evaluate a classifier?

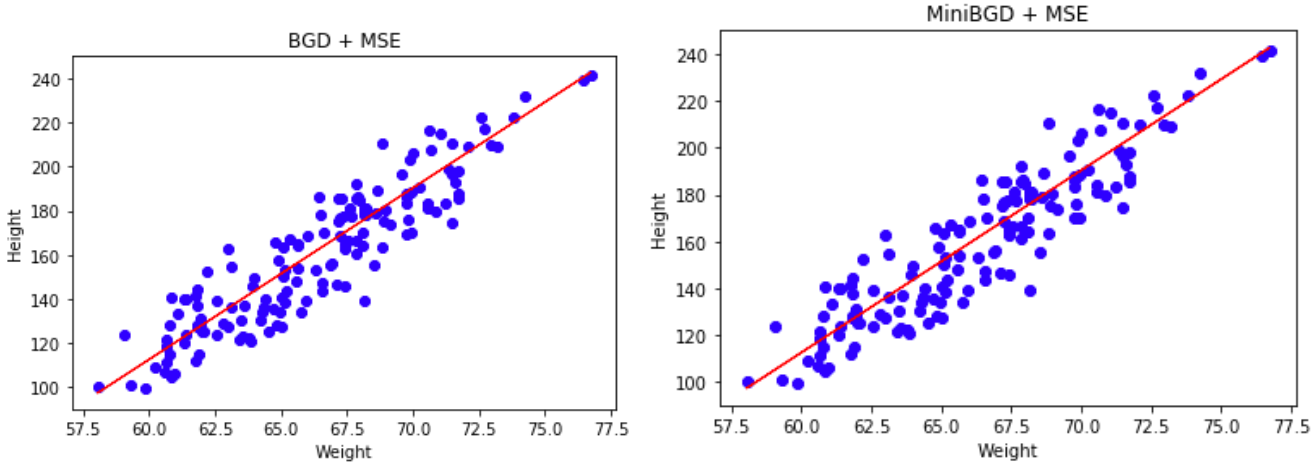
Accuracy is a good measure when the target variable classes in the data are nearly balanced. But in this dataset the classes is unbalance, accuracy is not the best performance metric to evaluate the classifiers. For example in this dataset there are 24719 people's incomes are less than 50k, only 7841 people's income are greater than 50k. Let's say our model is very bad and predicts every case as less than 50k. In this situation, 24719 people has classified currently and 7841 are not. Now even though the model is terrible at predicting incomes is less than 50k, The accuracy of such a bad model is also 76%. There are serval other performance metric better to evaluate the classifier. One of the performance metric is precision, Precision is a measure that the proportion of predict positives (people's income are greater/less than 50k) and the people's income actually have greater/less than 50k. Another measurement standard is recall, recall is measure for the proportion of the people that actually have greater than 50k income was classify as less than 50k income. The last metric is F1-score, F1-score is single score that kind of represents both precision and recall. The formula shows as follow: F1 Score = 2 * Precision * Recall / (Precision + Recall).

Best algorithms

According to each of the four performance metrics, I think the two of the best algorithms are AdaBoost and Gradient Boosting algorithm. They don't have the same performance metrics, but there are quite similar. First of all, the accuracy of AdaBoost and Gradient Boosting algorithm respectively are 0.85 and 0.86, they have the highest accuracy over all. If we look for the precision metrics, AdaBoost have the highest precision for classify income lower than 50k, and Gradient Boosting algorithm have the highest precision for classify the income greater than 50k. To choose which algorithm is more suitable for this dataset, I think is might depends on what goals you want to achieve. If you want to focus on classify the people actually have lower than 50k income, is better to use AdaBoost algorithm, but if you want to focus on the income greater than 50k, is better to use Gradient Boosting algorithm. Now we look at the recall performance: they both doing a good job on accurately retrieved income lower than 50k. But Gradient Boosting algorithms do not perform very well on retrieved income greater than 50k, it leads to pull down the performance on F1-score. In conclusion F1-score is the comprehensive evaluation of precision and recall, and both algorithm does achieve highest score in F1-score over all.

Part 3: Optimisation Methods

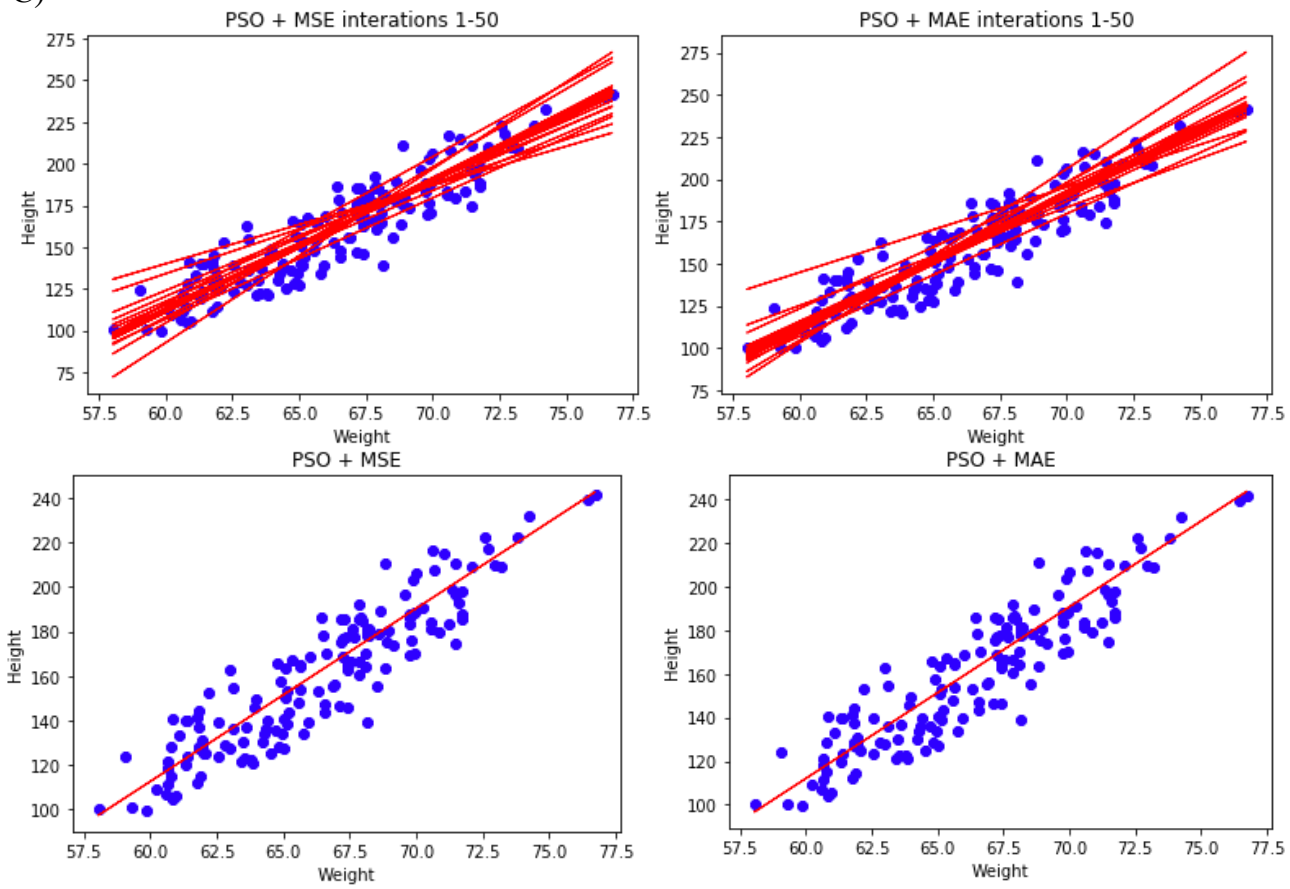
1 A)



B)

	Execute on Time	R-Squared	MSE	RMSE	MAE
BGD+MSE	0.076	0.81	163.56	12.78	10.67
MiniBatchBGD+ MSE	0.163	0.81	163.53	12.79	10.67
PSO+MSE	0.481	0.81	163.54	12.79	10.67
PSO+MAE	0.351	0.81	164.83	12.84	10.73

C)



D)

The fastest method is BGD+MSE, and the slowest method is PSO+MSE. In BGD algorithm the formula shows as follow:

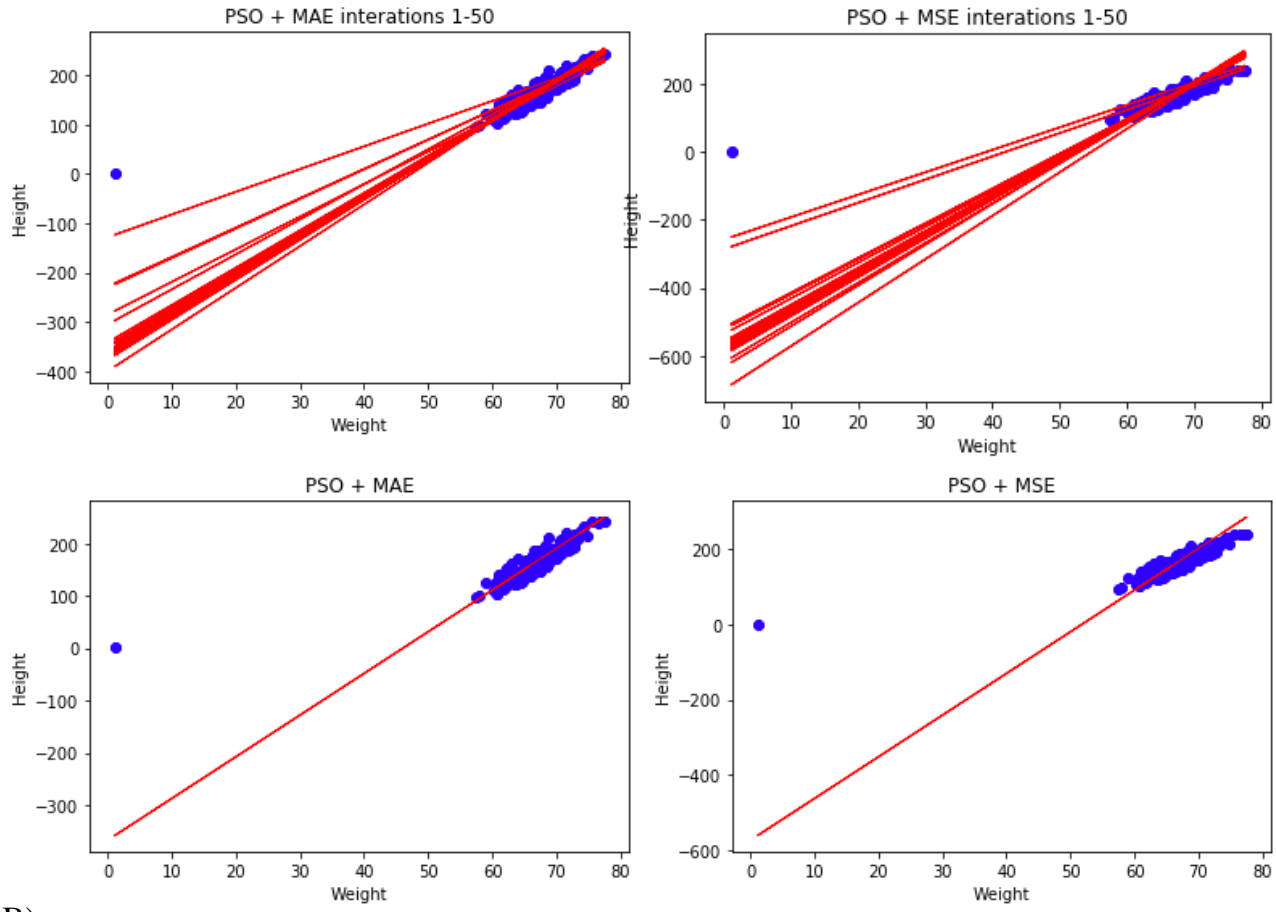
$$\theta_j^{'} = \theta_j + \frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$$

Since it is necessary to minimise the risk function, each of the theta is updated by the negative direction of the gradient of each parameter theta:It can be noticed from the above formula that it obtains a global optimal solution, but each step of the iteration uses all the data of the training set. If m is large, then iteration of this method can be imagined.The speed will be quite slow. But in this model the maximum interaction are only 50, so the model are training quite fast. By contrast, the PSO algorithm is more complex. The PSO is initialised as a group of random particles (random solutions). Then find the optimal solution by iteration. In each iteration, the particle updates itself by tracking

two "extreme values" (Pbest, Gbest). After finding the two optimal values, the particles update their speed and position by the following formula:

$$v_i = \omega \times v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i)$$

2 A)



B)

Compare the above two plots I draw, I find out PSO+MSE is less sensitive to outliers. Because during the model training, the metric type it use for calculate computer loss is mean square. Therefore the loss values return will impact by the outliers values. It will leads to the whole regression line slightly unfit for group of the values, only because this outlier. But MAE is different, it can perfectly avoid this problem. During the model training the loss values return is calculate by mean absolute. It can avoid the problem that the errors cancel each other out, so it can accurately reflect the actual prediction error. The above PSO + MAE figure shows the regression line perfectly fit with the group of the values.

C)

NO, we cannot use gradient descent or mini-batch gradient descent to optimise MAE. Because they are sensitive.