Comp309 Assignment 4

Jiawei Luo -300434604

Part 1: Performance Metrics in Regression

1: exploratory data analysis (EDA) :

Following the 7 step Below, The process and result of 'Diamonds' dataset was analyzed.

Step 1: Load data

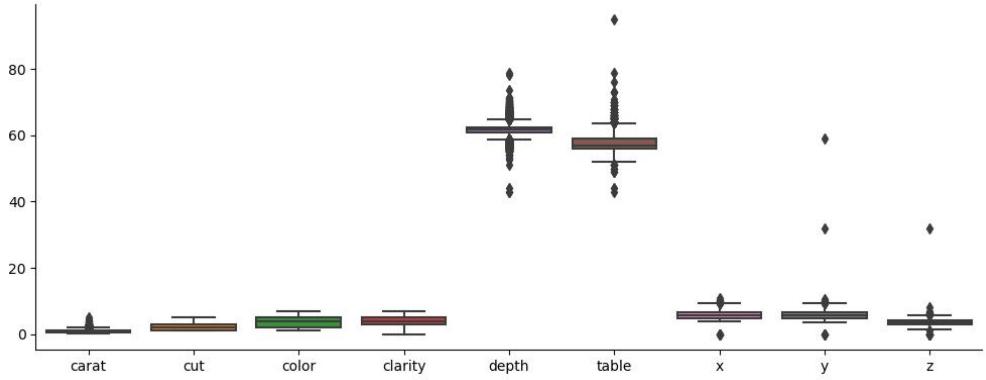
Using the module pandas and import as pd, The dataset was loaded by pd.read_csv() as The pandas Dataframe for further data exploration and analysis.

Step 2: Initial Data Analysis

Initially, the dataset has ten features and one label(Price), The goal of our exploration is to find the relationship between features and Prices, and predict the Price based on the features and trained model. The column 1 here is a unique attribute named index, and it would be deleted due to its uniqueness, also, it has no relation to the Price we are going to predict. Column 2 to column 10 indicates the nine features of each diamond. There are three nominal features which are "cut", "color" and "clarity" while the other features are numerical. Column 11 is the Price of the diamond, which is the label where we want to predict. Using the df.shape, we could know there are 53940 instances in the dataset, and the missing value was checked, there is no missing value in this dataset.

	index	carat	cut	color	clarity	depth	table	x	y	z	price
count	53940.000000	53940.000000	53940	53940	53940	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
unique	NaN	NaN	5	7	8	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	Ideal	G	SI1	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	21551	11292	13065	NaN	NaN	NaN	NaN	NaN	NaN
mean	26970.500000	0.797940	NaN	NaN	NaN	61.749405	57.457184	5.731157	5.734526	3.538734	3932.799722
std	15571.281097	0.474011	NaN	NaN	NaN	1.432621	2.234491	1.121761	1.142135	0.705699	3989.439738
min	1.000000	0.200000	NaN	NaN	NaN	43.000000	43.000000	0.000000	0.000000	0.000000	326.000000
25%	13485.750000	0.400000	NaN	NaN	NaN	61.000000	56.000000	4.710000	4.720000	2.910000	950.000000
50%	26970.500000	0.700000	NaN	NaN	NaN	61.800000	57.000000	5.700000	5.710000	3.530000	2401.000000
75%	40455.250000	1.040000	NaN	NaN	NaN	62.500000	59.000000	6.540000	6.540000	4.040000	5324.250000
max	53940.000000	5.010000	NaN	NaN	NaN	79.000000	95.000000	10.740000	58.900000	31.800000	18823.000000

In this case, the 'carat', 'depth' and 'table' gives some reasonable values with some outliers statistically. The mean are very close to the "50%" values, so they are symmetrically distributed. However, The 'x', 'y' and 'z' are three unknown features with some value at 0. They are features with same type(size) of information due to their names, one thing needs to figure out is that the minimum number of them are 0, which is not make sense in a physical definition of a diamond. They were simply defined as outlier. On the other hand, The range of depth and table is obviously larger then others, it may need to be scaled due to it can obscure the statistical significance of



the features and produce inaccurate coefficients.

Step 3:Preprocess Data

A) Convert data type

Because the regression model can not directly handle the ‘string’ type of data, which is nominal. It is necessary to convert the data type of ‘cut’, ‘color’ and ‘charity’ to be numerically represented. Generally, a good method of deal with nominal features is to using one-hot coding, which is getting dummies of the features and convert it to ‘1’ and ‘0’. it will increase the number of columns based on the number of elements in features. Because there are only a few elements here, I directly convert the nominal features to numbers:

‘cut’: Ideal':1,'Premium':2,'Very Good':3,'Good':4,'Fair':5

‘Color’: 'D':1,'E':2,'F':3,'G':4,'H':5,'I':6,'J':7

‘Clarity’: 'IF':0,'VVS1':1,'VVS2':2,'VS1':3,'VS2':4,'SI1':5,'SI2':6,'I1':7

B) Remove outliers

The outliers are defined as the points appear to be significantly different from other observations. An outlier may indicate bad data. Statistically, in the box plot above, there are many outliers out of the quantile + 1.5 IQR, they will be removed in later works.

C) Dataset split

The dataset was split randomly based on a random seed 309. The whole dataframe was split to 30% of test dataframe and 70% of train dataframe. In order to build the model and train it, both test and train dataframe were split to x_test, y_test and x_train, y_train, x is the features and y represents the ‘price’. After split dataset, there are 16182 instances in the test set and 37758 instances in the training set

D) Remove unique feature

The first column that represents the index of instances; it is the unique feature that all the instances have different numbers. It generally has no correlation to the label that we are going to predict, and sometimes it would become noise if we don’t remove this column.

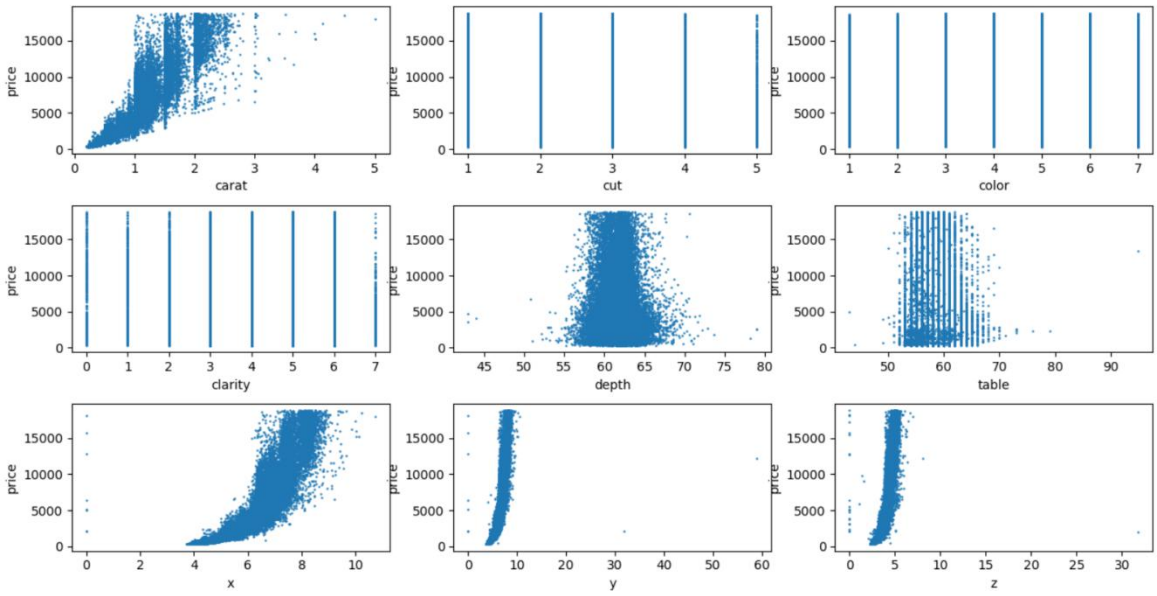
Step 4: Exploratory data analysis

A) The correlation coefficient of each features vs price

	carat	cut	color	clarity	depth	table	x	y	z
price	0.9216	0.0535	0.1725	0.1468	-0.0106	0.127	0.8844	0.8654	0.8612

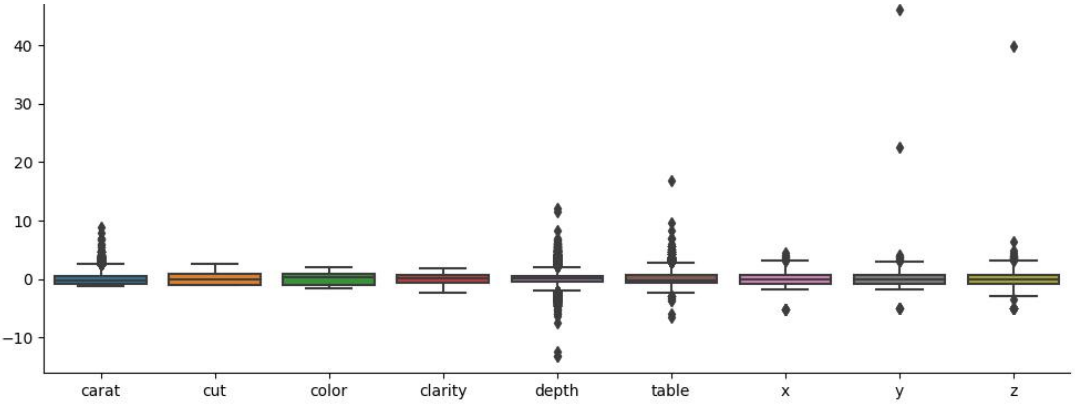
The correlation coefficient of the dataframe could be simply obtained by using df.corr(), As the result above, the ‘carat’, ‘x’, ‘y’, ‘z’ show strong positive relation to the price while the correlation coefficient of ‘depth’, ‘cut’ is quite close to 0, which show a poor correlation to the price. So the carat and the size of the diamond have a significant impact on the price of the diamond.

The scatter plot below shows each features against the price: For the three strong correlation features, which is ‘carat’, ‘x’, ‘y’, ‘z’, we can obviously see the trend that when these values increases, The y axis, which is the price of diamond, is increasing too. Meanwhile, it is hard to observe the correlation between the price and the low correlation coefficient features such as ‘cut’, ’depth’ and ‘table’.



B) Standardizing

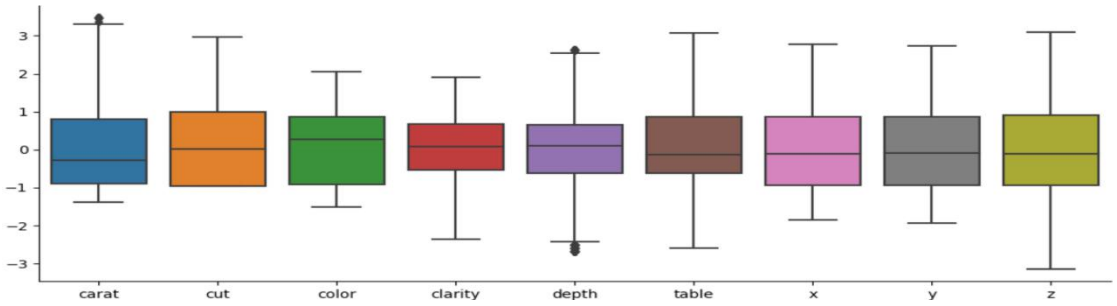
The dataframe of x_train and x_test were standardized using the standard scaler in preprocessing of Sklearn. It is the process of subtracting the mean and dividing by the standard deviation. After standardized, the data points were centred around 0, and it is noticeable that though standardizing, the range of original numerical data such as ‘depth’ and ‘table’ will not influence the value of coefficients when we do the regression. We cannot compare regular regression coefficients since they have different ranges, But we can compare the standardized coefficients to understand the importance of each feature as they used the same scale.



C) Remove outliers

One of my other data explorations is moving the outliers based on the statistical method, the outliers defined by box plot are the points out of the 1.5IQR of first Quartile and third Quartile. So after I remove the instances that have outliers, the box plots look like below:

The training set after remove outliers has 33335 instances, which means 4423 instances were removed. Since it significantly influences (or decline) the reliability of the model and dataset, I would not use it in my final model.



Step 5: Training the Model and testing

Techniques(default setting)	MSE	Rank	R^2	Rank	MAE	Rank	Execution time	Rank
Linear regression	1543588.13	6	0.9	5	810.46	6	0.0309	2
k-neighbors	549769.7	4	0.97	3	382.49	4	1.9888	6
Ridge	1543588.32	7	0.9	5	810.57	7	0.0089	1
decision tree	546336.62	3	0.97	3	357.7	3	0.2968	5
random forest	327420.72	1	0.98	1	281.24	1	2.3636	7
gradient Boosting	398356.08	2	0.98	1	343.0	2	2.4545	8
SGD	1565689.85	8	0.9	5	806.03	8	0.0459	3
support vector	8080037.63	10	0.5	10	1347.54	10	137.0381	10
Linear SVR	2534691.92	9	0.84	9	858.25	9	0.0769	4
multi-layer perceptron	718191.72	5	0.96	8	484.21	5	45.9407	9

The table is the results of each regression technique that be trained by the training set and applied to the test set. All the parameters of techniques were set to default initially. MSE, R2, MAE and RMSE were chosen to evaluate the prediction. RMSE basically is the same thing as MSE, so i did not add it to the table. The ranking of each technique in terms of MSE, R2 and MAE is also demonstrated.

The top two learning method in terms of MSE, MAE and R2 are random forest and gradient Boosting, while the performance of support vector regression was much lower than others, and it obtain the worst results in the ranking. In terms of execution time, Top 3 are Ridge, Linear regression and SGD, and They are all from the linear_model API. The support vector regression and multi-layer perceptron spent a long time on training, 137s and 45.9s respectively, the speed of them are very slow. It is noticeable that the random forest and gradient boosting are all from the ensemble, they got a really good result with slightly slower speed then linear model.

Tuning parameters

The tuned parameters of 10 learning method and their changes are show below

Learning method	Tuned Parameters	Describe	MAE Bef.	MAE Aft.	R2 Bef.	R2 Aft.
Linear regression	fit_intercept=True, normalize=False	calculate the intercept, and standardized before modelling	810.46	810.46	0.9	0.9

k-neighbors	n_neighbors=11,algorithm='auto',leaf_size=15, p=2	The larger k the better, but the promote was not obvious	382.49	378.19	0.97	0.97
Ridge	alpha=0.5, fit_intercept=True, normalize=False	Smaller alpha better, but not obvious	810.57	810.51	0.9	0.9
decision tree	criterion='mse',random_state=30	Random state slightly decreased the MAE	357.7	355.65	0.97	0.97
random forest	n_estimators=300	The larger number of estimator is better	281.24	268.83	0.98	0.98
gradient Boosting	learning_rate=0.02,n_estimators=1000, tol=1e-4	The larger number of estimator is better, but slower	343.0	319.15	0.98	0.98
SGD	max_iter=1000,tol=1e-3, early_stopping=True	Early stopping can prevent overfitting, not really work for small dataset, speed up training	806.03	803.03	0.9	0.9
support vector	C=1000.0,kernel='poly'	The large C have better result	1347.54	711.26	0.5	0.88
Linear SVR	C = 10.0,loss = 'squared_epsilon_insensitive',dual = True	The large C have better result, the loss function not influence too much(squared or not)	858.25	801.22	0.84	0.84
multi-layer perceptron	learning_rate_init=0.005,early_stopping=True	Small Learning rate have better result, but slower. Early stopping prevent overfitting of model	484.21	372.94	0.96	0.97

The Tuning of parameters of 10 learning method has many aspects to play with; most of them can have slightly changed when setting different parameters. Because the R2 was limited to 2 decimal, the change in R2 is usually not significant. It is noticeable that the MAE of random forest and gradient Boosting dropped with larger C value then default value, but the execution time are much slower than before. Similarly, after set larger C values in SVR and Linear SVR, the MAE can be dropped. Especially for the SVR, when the “Linear” Kernel was used, The R2 was significantly increased from 0.5 to 0.88, The MAE was also decreased a lot. Meanwhile, using early stopping in MLP could significantly promote the result from preventing overfitting of the model but slight in SGD. Because there are some random terms, some results probably vary from executions.

The results after tuned parameters show below:

Techniques(tuned parameters)	MSE	Rank	R^2	Rank	MAE	Rank	Execution time	Rank
Linear regression	1543588.13	7	0.9	6	810.46	9	0.014992	2
k-neighbors	546575.88	5	0.97	3	378.19	5	2.348647	5
Ridge	1543588.08	6	0.9	6	810.51	10	0.009994	1
decision tree	546587.52	4	0.97	3	355.65	3	0.324813	4
random forest	298104.66	1	0.98	1	268.83	1	72.32335	9

gradient Boosting	343002.37	2	0.98	1	319.15	2	25.3234	7
SGD	1553333.01	8	0.9	6	803.03	7	0.117924	3
support vector	1891008.49	10	0.88	9	711.26	6	323.7265	10
Linear SVR	1556721.07	9	0.84	10	808.55	8	8.029374	6
multi-layer perceptron	529385.78	3	0.97	3	372.94	4	28.1672	8

Compare with the results before tuning, The ranking of SVR in terms of MAE was increased a lot. The increased Rank of each term was marked by the red box, and the decreased Rank was marked by green box, while the while white boxes remain no change. In this table, the MLP and SVR achieved higher Rank then before in terms of MAE, MSE, RMSE and R2. However, the overall execution time was increased, especially for the random forest, gradient Boosting and SVR. The Top 3 learning methods that will be chosen in this case are random forest(achieved best performance), gradient Boosting (also good performance and slightly faster than random forest), multi-layer perceptron (slower speed, but great performance on regression problem, the performance could be further improved by tuning better parameters). The trade-off between execution time and the performance of the learning method should be considered in the selection of method when we are dealing with a large dataset.

Part 2: Performance Metrics in Classification

1) exploratory data analysis and preprocessing

In this part, two datasets were given. One is the training set ‘adult.data’, 32561 instances are contained in this training set. The other one is the test set ‘adult.test’, 16281 instances in the test set. Both datasets were stored by using two pandas DataFrame, but both datasets did not contain the name of features and label. Using the name derived from the UCI machine learning repository(https://archive.ics.uci.edu/ml/datasets/Adult), The features were named by feature_name = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss','hours-per-week', 'native-country', 'class'], So the feature names were used for column names.

so there are 14 attributes and one class that we need to classify. The predict task is to determine whether a person earns over 50K in a year, two classes are used: ‘>50K’ (‘>50K.’ in test set) was set to positive(‘1’) and ‘<=50K’(‘<=50K.’ in test set) was set to negative(‘0’).

Originally, missing values in both training set and test set were represented by ‘ ?’, to deal with missing value in dataframe, all the missing value (‘ ?’) were replaced by using ‘df.replace(" ?", value=np.NaN)’, In this case, missing values are represented by NaN of np. Too many missing values influences the reliability of model, so missing values were replaced by the most frequent value in each attributes for nominal features.

There are many nominal attributes in the training set and test set, which were represented by ‘string’. To handle string values, one hot coding(or dummy) should be used to convert those string values (nominal attributes)to binary. It basically converts the unique values in each feature to binary columns for each category. So after getting dummies, the training set has 105 columns while the test set has 104. There is a new problem that the dimension of two data frame is different, which means one unique term showed in the training set but not in test set. To fix this, one column (one element in ‘native country’, the ‘Holand-Netherlands’) has to be dropped using ‘train_features.drop(["native-country_ Holand-Netherlands"], axis=1)’.

Similar to part 1, To using the classifiers, labels and features should be used separately, A 'unlabeldata' function was also used to return features and class.

2) report the results (default setting)

R: Ranking

method	accuracy	R	precision	R	Recall	R	f1	R	AUC	R	Execution time	R
KNN	0.78	10	0.55	10	0.32	6	0.41	8	0.62	7	4.69	7
naive Bayes	0.8	6	0.64	8	0.3	7	0.41	8	0.63	6	0.66	1
SVM	0.8	6	0.97	1	0.15	10	0.26	10	0.58	10	158.53	10
DT	0.81	5	0.6	9	0.61	1	0.61	5	0.74	5	1.48	3
random forest	0.85	3	0.71	6	0.58	4	0.64	3	0.75	3	1.96	4
AdaBoost	0.86	2	0.77	4	0.6	2	0.67	2	0.77	2	3.87	6
Gradient Boosting	0.87	1	0.79	2	0.6	2	0.69	1	0.78	1	17.88	8
linear discriminant	0.84	4	0.72	5	0.56	5	0.63	4	0.75	3	2.08	5
MLP	0.8	6	0.78	3	0.24	9	0.36	7	0.61	8	25.76	9
logistic regression	0.8	6	0.7	7	0.26	8	0.38	6	0.61	8	0.66	1

Is accuracy the best performance metric to evaluate a classifier?

Accuracy is a good evaluation of classification, but the selection of evaluation should depend on the dataset. In this dataset, the class was highly unbalanced. in the training set, 24720 people's income is smaller than 50K, while 7841 are more than 50K. In the test set, 12435 people have income that smaller than 50K, and only 3846 people are more than 50K. if we consider a very bad model that predicts all the people in the test set have income smaller than 50K, The accuracy will still 76%, which is not too bad. However, we can not say that the model that predicts all the income smaller than 50K is good.

On the other hand, there are many other methods to evaluate the model rather than only using accuracy. Such as precision and recall, precision is calculated by the proportion of True Positive(the number of positive that be predicted) and the actual number of positive, while the recall(also called Sensitivity) is calculated from the number of positive that actually are positive(true positive) over the number of true positive plus the number of negative that actually are positive. By looking at precision and recall, the problem of high performance on accuracy but the number of actual negative is much more than positive could be solved. Furthermore, a combination of precision and recall, which called F1 score could be used to find the balanced classification model with best blend of precision and recall.

Meanwhile, AUC of ROC could be used, The closer the ROC curve is to the upper left, the higher the performance of the model. AUC is the area under the ROC curve, so 1 of AUC indicates a

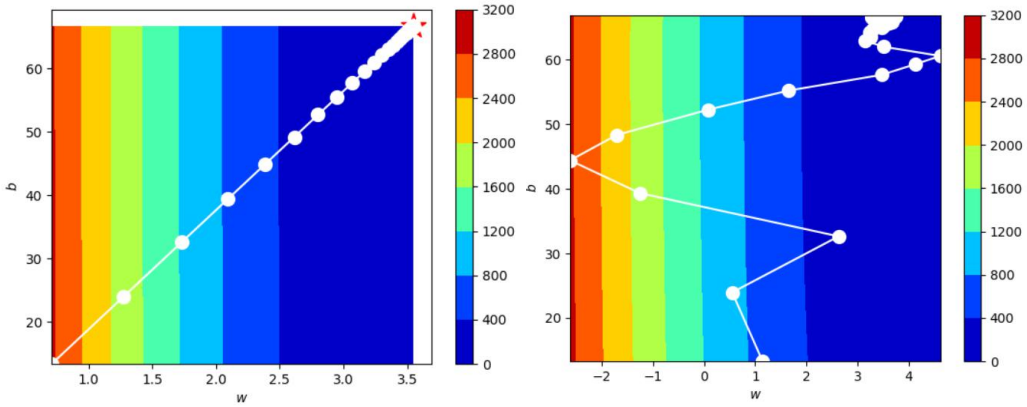
perfect classifier. Both AUC and ROC measure the overall performance of models.

According to each of the four performance metrics, The best algorithm is Gradient Boosting, while the second one is AdaBoost. Both of the two algorithms are derived from ensemble algorithms, which used multiple estimators and chose the best models. The accuracy of Gradient Boosting and AdaBoost is 0.87 and 0.86, respectively. The accuracy is very high. The F1 score and AUC score of both Gradient Boosting and AdaBoost are the best, which means the model of gradient boosting is well-balanced. Because they are all boosting learning method, the results of them are quite similar. It is noticeable that the method that has the best precision is SVM, and the Decision tree also obtains the highest recall score. SVM is good at predicting positive terms(predict the number of people who has income more than 50K) and decision tree can predict the True negative well(the number of people who has income less than 50K). but both methods have a bad F1 score, so they do not have a great balance of precision and recall.

Part 3:

1 running the dataset without outliers:

A) paths of gradient descent of BGD+MSE and MiniBatchBGD+MSE



The path of BGD+MSE is a straight line, The batch gradient descent calculates the error for each data in the training set, but updates the model only after all the training data have been computed. The BGB+MSE has a stable error gradient and a stable convergence point can be obtained. But, the stable error gradients may lead to premature convergence of the model to a parameter set that is not an optimal solution, also, the training speed is very slow if training set is very large.

The path of MiniBatchBGD+MSE is highly unstable and have very high variance. The training set is divided into several batches by Mini batch gradient descent, and the errors are calculated for each batch and the parameters are updated. The update frequency which decreases faster than the batch gradient is advantageous to the more robust convergence and avoids the local optimization. Also, we don't have to put all data in the memory. The speed of MiniBatchBGD is faster than BGD.

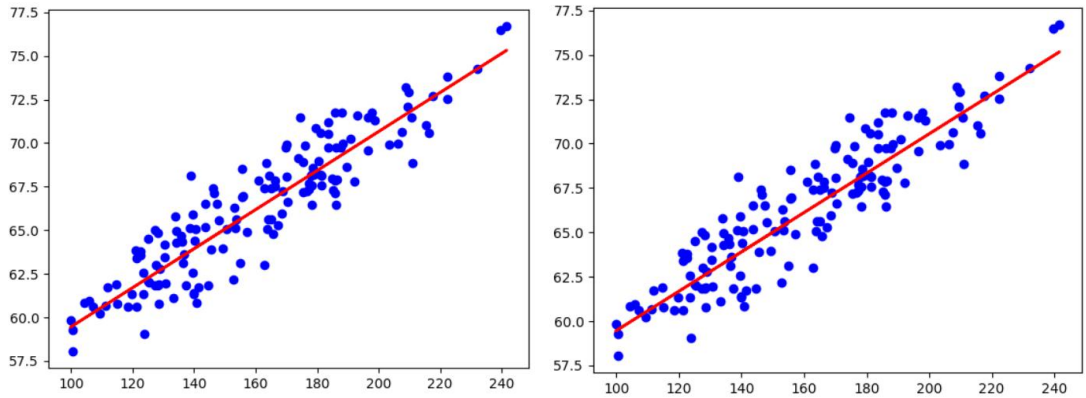
B)

	MSE	R-Squared	MAE
BGD+MSE	2.41	0.83	1.28
MiniBatchBGD+MSE	2.62	0.82	1.32
PSO+MSE	2.41	0.836	1.279
PSO+MAE	2.428	0.835	1.284

The table above shows that the model of MiniBatchBGD+MSE has the highest MSE value and MAE value, but it has the smallest R-Squared value, So the performance is poor. The model of PSO+MSE obtained the best R-Squared value, and the smallest MSE and MAE, so it has the best performance in four models.

C)

a scatter plot with the regression line learnt by PSO+MSE and PSO+MAE and the data points in the test set



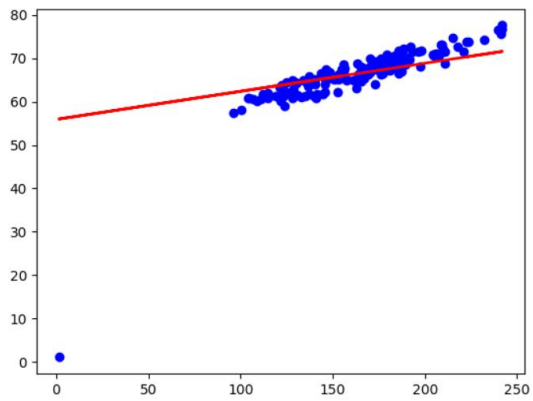
D)

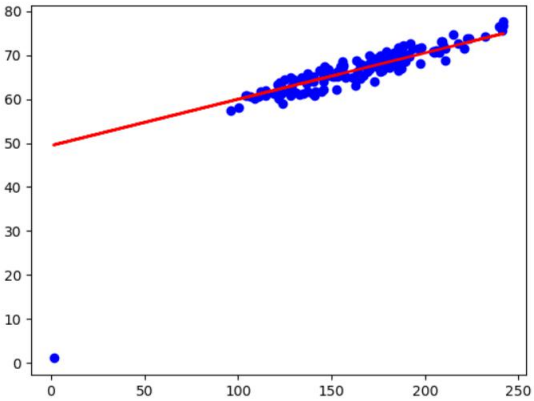
Algorithm	Execution time(second)
BGD+MSE	0.004
MiniBatchBGD+MSE	0.023
PSO+MSE	1.082
PSO+MAE	0.558

In this case, The BGD+MSE is the fastest one in terms of execution time while the PSO+MSE is the slowest algorithm. MiniBatchBGD is slower than BGD in this case, because the dataset is relatively small, The speed of MiniBatchBGD is commonly faster than BGD when dataset is very large and may not even fit into the computer's memory. Another reason could be the mini batch size was not optimized, while the algorithm of BGD was optimized. The Particle swarm optimization is the slowest because there are more computation steps and cycles in the Particle swarm optimization, and because the exponential is slower than the subtraction, the Particle swarm optimization converges slowly during the iteration.

2 running PSO+MSE, and PSO+MAE dataset with outliers

A) scatter plot of the regression line and the data points, PSO+MSE, and PSO+MAE respectively.





B)Compare with the scatter plot with outliers, The performance of two models that trained by the dataset without outliers are quite similar, The two lines are very close. The regression line of PSO+MSE is more biased by outliers, while the PSO+MAE is less sensitive to outliers. The reason could be that in the process of calculating the MSE, The errors were squared, But the errors were not squared in MAE. So Errors have higher weights in the MSE, which makes MSE is more sensitive to outliers.

C) We can not use BGD or mini-BGD to optimise MAE. The cost function of MAE is linear, which means the change of each adjustment of each loop is the same when we doing the converging. The converging is more sensitive than MSE, whose cost function is parabolic, the learning rate can be adjusted automatically in the optimization process. Even we use small learning rate to optimize MAE, the converging may not be precise, and the final model may not be as accurate as the MSE.