Assignment #4

Performance Metrics and Optimisation

COMP 309 Machine Learning Tools and Techniques

Qiangqiang Li(Aaron)

ID: 300422249

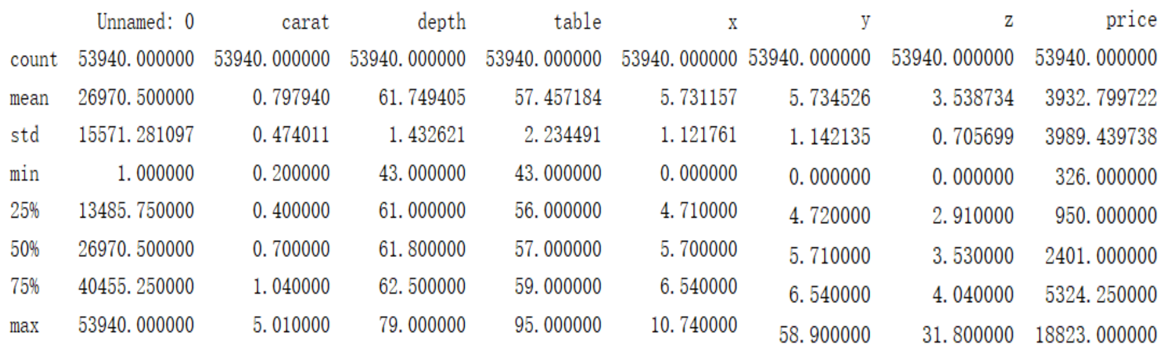
Part 1: Performance Metrics in Regression

Exploratory Data Analysis

- **Step 0 - Parameters Setting:** In this step, several parameters were set in the beginning of the program. Random seed and test split size were set to **309 and 0.3**, respectively.

- **Step 1 - Load Data:** The diamonds.csv was read into memory as a Data Frame using read_csv() for further data processing and analysis.

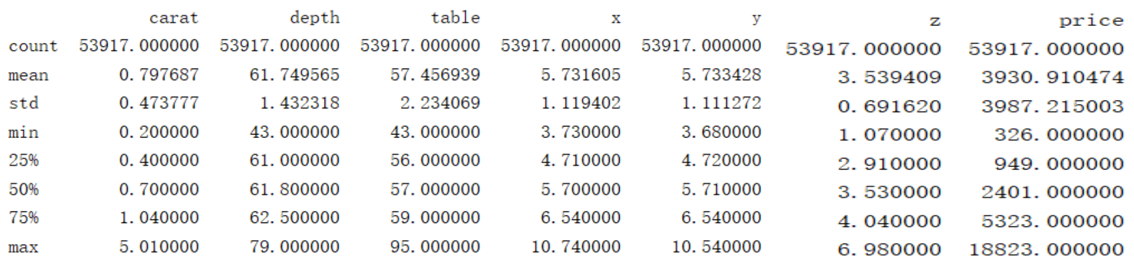
- **Step 2 - Initial Data Analysis: Firstly,** the first and last 5 rows of raw data in diamonds.csv were printed. These data were including 11 columns of data in total. **Column 2 to column 10 record** the 9 features of each diamond; **column 11 records the price**; **column 1 is the index column**, which can be removed later. By printing the shape of the Data Frame, it showed the raw data has **53940 instances**. Missing data was checked, and there was **no missing data in this case.**
  If print the statistical description and data type information of the raw data:



```
         Unnamed: 0         carat         depth         table             x            y             z         price
count  53940.000000  53940.000000  53940.000000  53940.000000  53940.000000 53940.000000  53940.000000  53940.000000
mean   26970.500000      0.797940     61.749405     57.457184      5.731157     5.734526      3.538734   3932.799722
std    15571.281097      0.474011      1.432621      2.234491      1.121761     1.142135      0.705699   3989.439738
min        1.000000      0.200000     43.000000     43.000000      0.000000     0.000000      0.000000    326.000000
25%    13485.750000      0.400000     61.000000     56.000000      4.710000     4.720000      2.910000    950.000000
50%    26970.500000      0.700000     61.800000     57.000000      5.700000     5.710000      3.530000   2401.000000
75%    40455.250000      1.040000     62.500000     59.000000      6.540000     6.540000      4.040000   5324.250000
max    53940.000000      5.010000     79.000000     95.000000     10.740000    58.900000     31.800000  18823.000000
```

According to the result, there are 6 numerical features columns were summarized statistically. The data in **carat, depth, table and price** columns seem reasonable by initial observation. However, there are some findings in x, y and z columns. Even though there is no specific recording of their meanings, by observing the data and their names (x, y and z), these three columns probably record some dimensional information of the diamonds. So, the minimum of 0 in x, y and z seem suspicious. And the maximum of **58.9 in y and 31.8 in z**, comparing to the **75% values of y and z**, respectively, may indicate they are outliers. Before cleansing the data, EDA must be done to confirm the hypothesis.

- **Step 3.1 - Data Pre-processing (Data Cleansing):** In this step, some instances with unreasonable values or outliers in certain features were removed. The rationality of the conditions was explored in later steps. However, after finalizing the conditions, it is memory-efficient to put this step here, especially for big data. (1. drop the index column,2. clean the outliers). The result is show that the instances is **53917, So After data cleansing, 23 instances were removed.**



```
              carat         depth         table             x             y              z          price
count  53917.000000  53917.000000  53917.000000  53917.000000  53917.000000   53917.000000   53917.000000
mean       0.797687     61.749565     57.456939      5.731605      5.733428       3.539409    3930.910474
std        0.473777      1.432318      2.234069      1.119402      1.111272       0.691620    3987.215003
min        0.200000     43.000000     43.000000      3.730000      3.680000       1.070000     326.000000
25%        0.400000     61.000000     56.000000      4.710000      4.720000       2.910000     949.000000
50%        0.700000     61.800000     57.000000      5.700000      5.710000       3.530000    2401.000000
75%        1.040000     62.500000     59.000000      6.540000      6.540000       4.040000    5323.000000
max        5.010000     79.000000     95.000000     10.740000     10.540000       6.980000   18823.000000
```

- **Step 3.2 - Data Pre-processing (Categorical Feature Quantification):** There are 3 columns of features in the raw data were categorical, which are the famous 3Cs when customers selecting a diamond: cut, colour and clarity. In this step, the original ratings were replaced by scores. A scoring system from 100 downwards linearly was set up(binary to numeric):

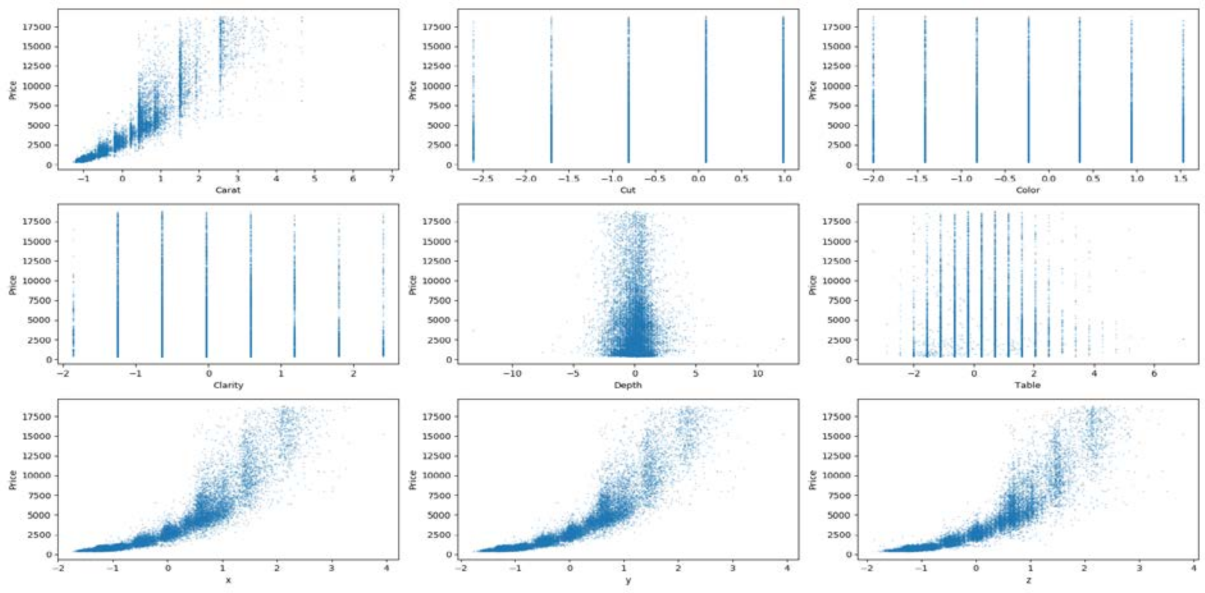
✧ **Cut: 'Ideal'**:100, **'Premium'**:90, **'Very Good'**:80,**'Good'**: 70,**'Fair'**:60

✧ **Clarity:'IF'**:100,**'VVS1'**:90,**'VVS2'**:80,**'VS1'**:70,**'VS2'**:60,**'SI1'**:50,**'SI2'**:40,**'I1'**:30

✧ **Color: 'D'**:100, **'E'**: 90, **'F'**:80, **'G'**:70, **'H'**:60,**'I'**:50,**'J'**:40

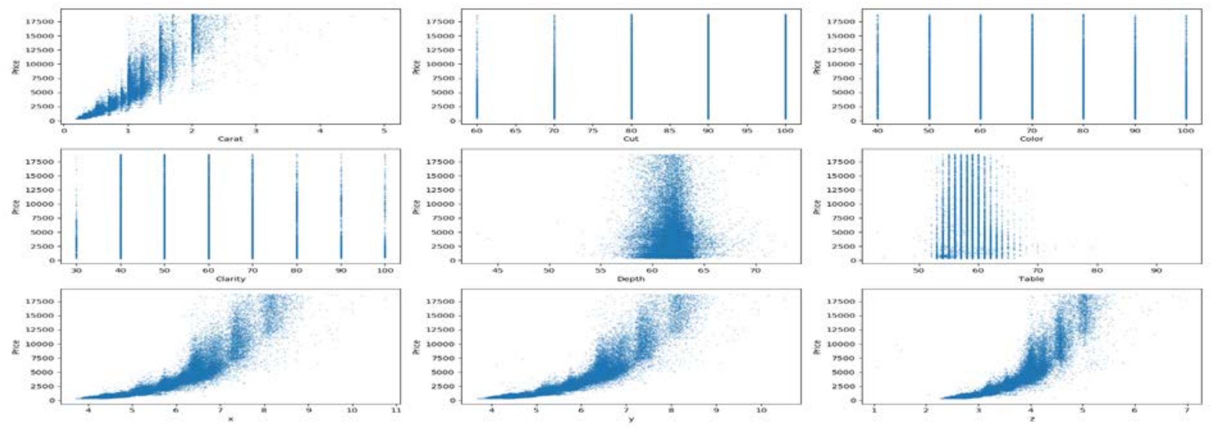
- **Step 3.3 - Data Pre-processing (Train-Test Split):** The data set was split into 2 DataFrames and 2 Series, Xs_train_set, Xs_test_set, y_train_set, y_test_set, with the shape of (37758, 9), (16182, 9), (37758, 1) and (16182, 1), respectively. The test_size was set in the beginning of the program: 0.3.

- **Step 3.4 - Data Pre-processing (Data Standardization):** The Xs_train_set and Xs_test_set were standardized by the **mean and STD** values of Xs_train_set. After data cleansing and standardization, the distributions are plotted below. It can be found clearly that the outliers were removed by comparing the plots below and the plots above.



- **Step 4 - Exploratory Data Analysis:** The correlation between the features was explored. From the result, we can see the carat and size of the diamond have a strong correlation with the selling price and the result of correlation between the price and other columns is shown below:

	carat	*cut*	*color*	*clarity*	*depth*	*table*	*x*	*y*	*z*	*price*
price	0.92	-0.05	-0.17	-0.14	-0.01	0.12	0.88	0.88	0.88	1.00



- **Step 5 - Modelling:** 10 regression algorithms were applied to the training data, and the models trained were used to predict the price in the test split. 4 Statistical values were selected in the performance metrics to evaluate the performance of each algorithm: R2, MSE, RMSE and MAE. The ranking in each column were added next to each one of them in italic font.

(Default Setting)	R2	R	RMSE	R	MAE	R	MSE	R	Execution Time(s)	R
LinearRegression	0.91	6	1199.23	6	800.34	6	1438141.63	6	0.0175	2
KNeighborsRegressor	0.97	3	711.74	3	378.08	4	506573.28	3	0.7015	6
Ridge Regression	0.91	6	1199.32	7	800.42	7	1438362.75	7	0.0045	1
DecisionTreeRegressor	0.97	3	740.13	5	362.06	3	547788.14	5	0.1834	5
RandomForestRegressor	**0.98**	**1**	**564.91**	**1**	**283.61**	**1**	**327212.81**	**1**	**1.0674**	**7**
GradientBoostingRegressor	**0.98**	**1**	**616.60**	**2**	**345.53**	**2**	**380189.63**	**2**	**1.1792**	**8**
SGDRegressor	0.91	6	1206.13	8	806.36	8	1462142.32	8	0.0506	4
SVR	**0.51**	**10**	**2783.86**	**10**	**1314.40**	**10**	**7749857.47**	**10**	**59.8324**	**10**
LinearSVR	0.85	9	1530.28	9	836.78	9	2341745.63	9	0.0236	3
MLPRegressor	0.97	3	738.33	4	417.41	5	545133.66	4	21.2900	9

The top 3 algorithms in terms of **R2, RMSE, MAE and MSE performance** are **Random Forest Regressor, Gradient Boosting Regressor and K Neighbors Regressor** (the first two are both ensembles learning methods). The worst performed algorithm is **Support Vector Regressor.**

If comparing **the execution time**, the ranking changes. The fastest 3 algorithms: **Ridge Regression, Linear Regression, and Linear SVR Regression**. Besides, **SGDRegressor, Decision Tree, K-Neighbors Regression and Ensembles** are slower, which take 10 times, 36 times and 140 times the average time Ridge Regression take. **Multi-layer Perceptron regressor and Support Vector Regressor** take the longest time, which is 4200 times and 14000 times longer than the Ridge Regression models.

Parameters Tuning: tune any parameter(s), report 10 algorithm(s), which parameter(s) and the parameter value(s), like the below tables:

Algorithm	Tunable Parameters Count	Parameters Tuned	Best Parameters Combinations	R^2 using Default Setting	R^2 after Tuning
Linear Regression	4	fit_intercept normalize	fit_intercept = True	0.91	0.91
k-Neighbors Regression	8	n_neighbors weights leaf_size p metric	n_neighbors = 12 weights = 'distance' leaf_size = [5, 6, 7, 8, 9] p = [1, 2] metric = [minkowski, manhattan]	0.97	0.97
Ridge Regression	8	alpha fit_intercept normalize	fit_intercept = True normalize = False	0.91	0.91
Decision Tree Regression	12	criterion presort	Criterion = 'mse' Presort = False	0.96	0.97
Random Forest Regression	16	n_estimators criterion oob_score	n_estimators = 18 criterion = 'mse' oob_score = False	0.98	0.98
Gradient Boosting Regression	19	loss learning_rate n_estimators max_depth criterion alpha	Loss = 'ls' learning_rate = 0.02 (worse if larger or smaller) n_estimators = 1000 (the larger the better) max_depth = 5 (the larger the worse) criterion = 'mse'	0.98	0.98
SGD Regression	17	loss penalty alpha l1_ratio epsilon learning_rate	loss = 'squared_loss' penalty = 'l2' alpha = 0.0001 l1_ratio = 0.25 epsilon = 1 learning_rate = 'constant'	0.91	0.91
Support Vector Regression	11	shrinking C coef0 kernel	Kernel = 'linear' C = 500.0 (the larger the better)	0.51	0.89
Linear SVR	10	C loss dual	C = 5.0 loss = 'squared_epsilon_insensitive' dual = True	0.85	0.91
Multi-layer Perceptron Regression	21	activation solver learning_rate	activation = relu solver = lbfgs learning_rate = adaptive	0.96	0.98

The *Parameters Tuning of* result like the below table:

(***Parameters Tuning***)	R2	R	RMSE	R	MAE	R	MSE	R	Execution Time(s)	R
LinearRegression	0.91	6	1199.23	6	800.34	6	1438141.63	6	0.0090	1
KNeighborsRegressor	0.97	4	640.67	4	325.85	4	410457.37	4	0.7142	5
Ridge Regression	0.91	6	1199.32	7	800.	7	1438362.75	7	0.0093	2

					42					
DecisionTreeRegressor	0.97	4	740.48	5	360.77	5	546847.88	5	0.2344	4
RandomForestRegressor	0.98	1	550.72	2	275.44	2	303293.40	2	0.7201	6
GradientBoostingRegressor	0.98	1	524.33	1	271.64	1	274924.69	1	23.4949	10
SGDRegressor	0.91	6	1216.19	9	823.90	8	1479121.43	9	0.0240	3
SVR	0.89	10	1317.13	10	696.16	10	1734834.72	10	10.76755	8
LinearSVR	0.91	6	1199.41	8	797.22	9	1438582.00	8	3.8992	7
MLPRegressor	0.98	1	567.81	3	314.77	3	322405.80	3	13.591377	9

According to the comparing, after using the parameter optimization, the **R2 results** for **Multi-layer Perceptron Regression** only was improved significantly. The execution time of **Ridge Regression, Linear SVR, Gradient Boosting Regression** was increased. If comparing the performance between these regression methods, **Random Forest Regression, Gradient Boosting Regression, and MLP Regression** achieved a higher ranking in R2, MSE, MAE, and RMSE. If talking about execution time, the improvement of R2 of **Linear SVR and Gradient Boosting Regressor** cost by longer execution time.

Among all these algorithms, **MLP Regressor** is the only one performs better and ranks up in every performance index after optimisation.

Ranking	*R^2*		*MSE*		*MAE*		*RMSE*		*Execution Time(s)*	
Optimization	bef	aft	bef	aft	bef	aft	bef	aft	bef	aft
Linear Regression	6	6	6	6	6	6	6	6	2	1
KNeighbors Regressor	3	4	3	4	4	4	3	4	6	5
Ridge Regression	6	6	7	7	7	7	7	7	1	2
Decision Tree Regressor	4	5	5	5	3	5	5	5	5	4
Random Forest Regressor	1	1	1	2	1	2	1	2	7	6
Gradient Boosting Regressor	1	1	2	1	2	1	2	1	8	10
SGD Regressor	6	6	8	9	8	8	8	9	4	3
SVR	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**8**
Linear SVR	9	9	9	8	9	9	9	8	3	7
MLP Regressor	4	1	4	3	5	3	4	3	9	9

Green - Ranking up; Blue - Ranking drops

Part 2: Performance Metrics in Classification

1. **Based on exploratory data analysis, discuss what preprocessing that you need to do before classification, and provide evidence and justifications.**

- **Step 0 - Parameters Setting:** In this classification task, adult.data and adult.test are given separately. Random seed was set **309.Found the adult.test**, need to **delete the first row**:**|1x3 Cross validator;**
- **Step 1 - Load Data:** The adult.data and adult.test were read into memory as a DataFrame using

read_csv() for further data processing and analysis. According to the information from the website(**https://archive.ics.uci.edu/ml/datasets/adult**), the columns were showed the below array(columns):

columns = ["Age", "Workclass", "Final_Weight", "Education", "Highest_Grade", "Marital_Status", "Occupation", "Relationship", "Race", "Sex", "Capital_Gain", "Capital_Loss", "Hours_per_Week", "Native_Country", "Income_Class"]

- **Step 2 - Initial Data Analysis**

According to the results of DataFrame.tail() and DataFrame.shape, **there are 32560** rows of data in training set, **16280** rows of data in test sets. The **column numbers are 15** for both by checking the shapes of them. And there is **no missing data** in both data sets.

When using each column of **Data Frame. unique ()** to print **the unique value** for each column, there were some problems found:

1. **There are so many unique values in Final_Weight column.**

Final_Weight : [83311 215646 234721 ... 34066 84661 257302]

From the searching result, the weight for a responding unit in a survey data set is an estimate of **the number of units** in the target population that the responding unit represents.

According to achieve the aim that find a person in certain conditions in certain features can earn higher than $50k or not, the highly discrete values in **Final_Weight column** which may induce high noise for **the classifying process**, this column was deleted from both DataFrames.

2. **There is 1 respondent in training set but no one in test set comes from Holland**

Native_Country : [' United-States' ' Cuba' ' Jamaica' ' India' **' ?'** ' Mexico' ' South'…….]

Because the number is also small, we can either delete the instance in training set, or modify the country information to **'?'**, to make sure the training set and test set have the exact same columns.

3. **The Education column was found to have the same meaning as Highest_Grade**

Education: [' Bachelors' ' HS-grad' ' 11th' ' Masters' ' 9th' ' Some-college' ' Assoc-acdm' ' Assocvoc' ' 7th-8th' ' Doctorate' ' Prof-school' ' 5th-6th' ' 10th' ' 1st-4th' ' Preschool' ' 12th']

Highest_Grade: [13 9 7 14 5 10 12 11 4 16 15 3 6 2 1 8]

The Education column should be removed, because the Highest_Grade column has quantified unique values as the same mean as the education. When the highest grade become higher, it does mean the person has a better education background.

4. **The separate Husband and Wife in Relation repeats the information in Sex**

Relationship : [**' Husband'** ' Not-in-family' **' Wife'** ' Own-child' ' Unmarried' ' Other-relative']

Sex : [' Male' ' Female']

So, we can use the Husband and Wife in Relation column was combined as Husband-wife, and the information in Sex column can separate them.

5. **The column Income_Class can use the binary to easily classify.**

Income_Class : [' <=50K' ' >50K']

Income_Class was binarize to quantitative information **1 and 0**, which represent **'>50K' and '<=50K',** respectively. Also, dummy variables were created for all the categorical data. Till now, all the data in the data sets are quantitative. To make sure the data can work well in different algorithms, the training set and test set were standardized.

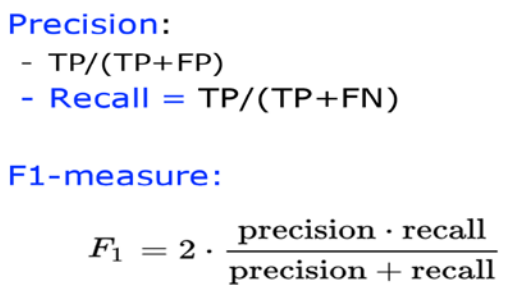
2. **Please report the results (keep 2 decimals) of all the 10 classification algorithms on the given test data in terms of classification accuracy, precision, recall, F1-score, and AUC. You should report them in a table.**

Using default setting for each classification algorithm, the accuracy, precision, recall rate, F1 score, and AUC, together with the rankings, are summarized below:

(Default Setting)	Acc	*R*	Prec	*R*	Rec	*R*	F1	*R*	AUC	*R*
KNeighborsClassifier	0.82	*8*	0.64	*8*	0.57	*10*	0.60	*9*	0.74	*9*
GaussianNaiveBayes	0.52	*10*	0.32	*10*	0.95	*1*	0.48	*10*	0.67	*10*
SVMClassifier	0.85	*3*	0.74	*3*	0.58	*7*	0.65	*5*	0.76	*5*
DecisionTreeClassifier	0.82	*9*	0.62	*9*	0.60	*5*	0.61	*8*	0.74	*8*
RandomForestClassifier	0.84	*7*	0.69	*7*	0.58	*8*	0.63	*7*	0.75	*7*
AdaBoostClassifier	0.86	*2*	0.75	*2*	0.61	*3*	0.67	*2*	0.77	*2*
GradientBoostingClassifier	0.87	*1*	0.80	*1*	0.61	*4*	0.69	*1*	0.78	*1*
LinearDiscriminantAnalysis	0.85	*6*	0.71	*5*	0.58	*9*	0.64	*6*	0.75	*6*
MultilayerPerceptronClassifier	0.85	*5*	0.70	*6*	0.63	*2*	0.66	*3*	0.77	*3*
LogisticRegression	0.85	*4*	0.73	*4*	0.59	*6*	0.65	*4*	0.76	*4*

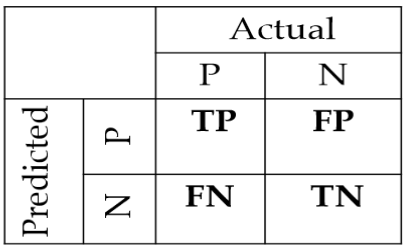
In the training data set, the value of the predicted feature (Income_Class) for **76%** of the instances are 0 (<=50K), **24%** are 1 (>50K), which is imbalanced.

3. **Is accuracy the best performance metric to evaluate a classifier? and why?**


Precision:
- TP/(TP+FP)
- Recall = TP/(TP+FN)

F1-measure:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$



		Actual	
		P	N
Predicted	P	**TP**	**FP**
	N	**FN**	**TN**

Reply: Accuracy **is not the best** performance metric to evaluate the performance of a classification. Because Like this case, **the data is imbalanced,** the classification accuracy alone **cannot be trusted** to select a well-performing model.

Comparing to accuracy, **AUC** seems better to evaluate the performance of a classification in this case. The higher the better. If PRC (Precision-Recall) can also be checked, that would prevent the superficial high performance in case the number of negative instances is much larger than positive instances. F1-score can reflect the overall consideration between precision and recall. However, due to its definition, when **the precision is high, recall is low**. So, for some cases, it is better to **investigate detail in** both **precision and recall** after evaluating F1-score.

4. **Find the two best algorithms according to each of the four performance metrics, Are they the same? Explain why.**

Reply: According to the result, the best-performance algorithm in terms of the performance metrics is **Gradient Boosting Classifier;** the second one is **Adaptive Boosting**. Both two algorithms belong to Boosting algorithm. Boosting is an ensemble method for improving the model predictions of any given learning algorithm, and combines some average-performed models to get a better-performed model.

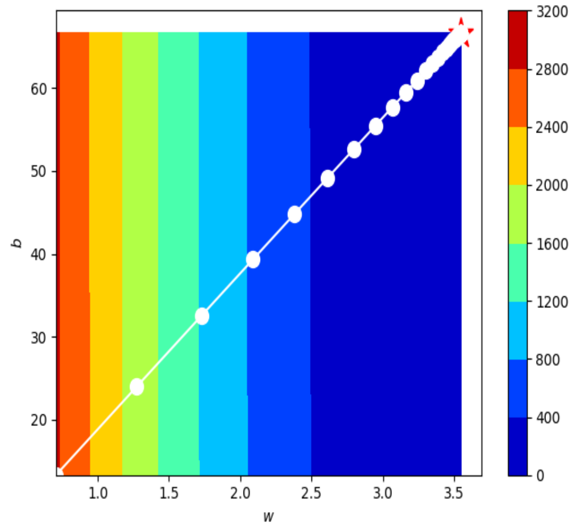
Looking at **Adaptive Boosting algorithm**, several linear classifiers are used to classify the data one by one. After each sub-classification step, the weights of the wrongly classified data are increased. Finally, all these linear classifiers are combined according to their weights. The weight of a certain sub linear classifier depends on its classification performance. And because every new sub linear classifier in sub-classification steps adjust based on the performance of the previous sub-model, this is reason why this Boosting algorithm is Adaptive.

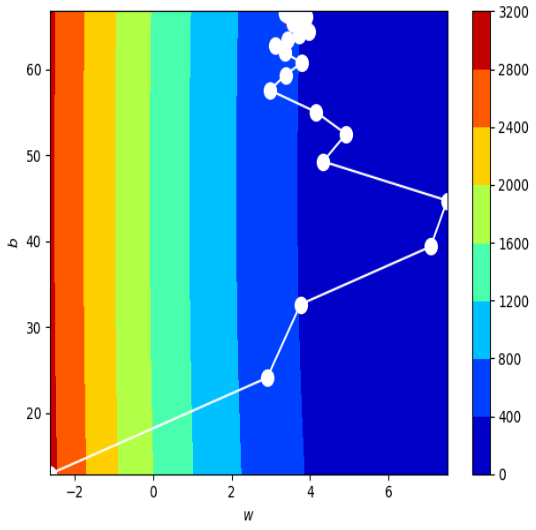
Gradient Boosting is a more general Boosting algorithm than Adaptive Boosting. Gradient Boosting combines the advantages of both Boosting and Gradient Descent. **The difference** between Adaptive Boosting and Gradient Boosting algorithms is that **Adaptive Boosting** improves the performance of a model by **increasing the weights of the wrongly classified data points;** however, the Gradient Boosting improves the model by **keeping calculating the gradient**, and **optimising the model toward the gradient descending direction**.

Part 3: Optimisation Methods

1. **On the dataset without outliers, i.e. Part2.csv. Run each of the BGD+MSE, MiniBatchBGD+MSE, PSO+MSE, and PSO+MAE methods on the training data to learn a linear regression model and test the learnt model on the test data.**

a) **Plot the paths of gradient descent of BGD+MSE and MiniBatchBGD+MSE, then discuss their differences and justify why.**





Reply:

Batch GD computes the gradient of the cost/objective function with the entire training dataset, to perform just one update. It is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. But the problems are the updating speed can be very slow, is intractable for datasets that do not fit in memory, and does not allow us to update our model online.

SGD considers only one more new data point in the weights updating process. This can speed up the training very much, and It can be used to learn online. However, SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

Mini-batch gradient descent is taking the best of both batch and SGD, and performing an update for every mini-batch of n training example. It reduces the variance of the parameter updates, and can lead to more stable convergence. Besides, it can make use of highly optimised matrix optimisation common to deep learning libraries that make computing the gradient very efficiently. But the problem is that Mini-batch gradient descent need to set the mini-batch size hyperparameter.

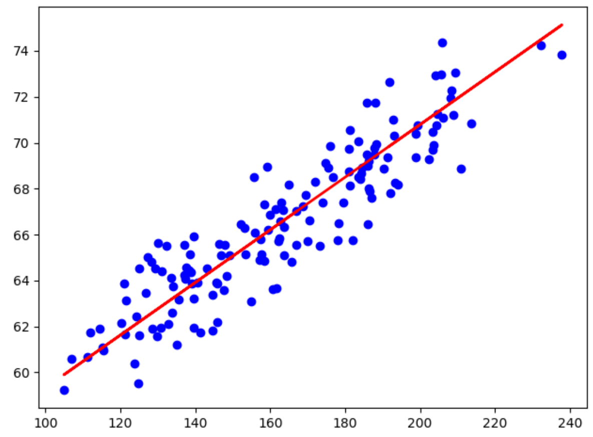
According to the two graphics, Let's see how it performs with a min-batch size of 10 samples, mini-batch gradient descent is "kicking" the cost function out of local minimum values to reach better, perhaps even the best, minimum. So **mini-batch gradient descent** appears be the superior method of gradient descent to be used in neural networks training.

b) Report the results (keep 2 decimals) of the four learnt models over the MSE, R-Squared, and MAE performance metrics on the test set. Compare their results and discuss the differences. You can report them in a table.

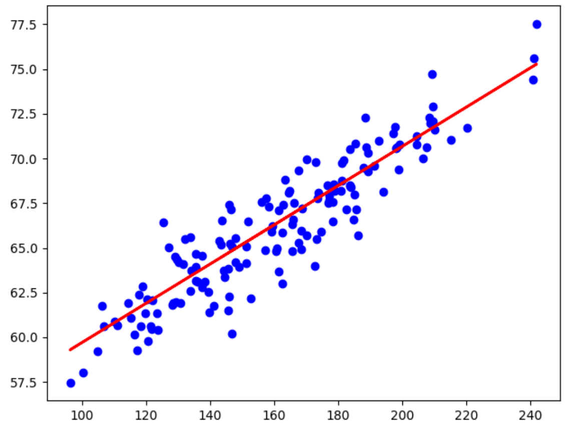
Metric Type	Optimizer Type	R2	MSE	MAE
MSE	BGD	0.84	2.41	1.28
MSE	MiniBGD	**0.85**	**2.01**	**1.12**
MSE	PSO	**0.86**	2.04	1.12
MAE	PSO	0.82	2.57	1.31

Reply: By comparing the performance metrics of the 4 models, **the best rate of R2** is **MSE+PSO**, and **MSE+MiniBGD and MSE+BGD are the middle rate**. On the other hand, **in the MSE and MAE performance metrics**, the **MSE+ MiniBGD** has **the lowest value** in the four models, which means it is the best model among these four models. While the **PSO+MAE** has the poorest performance, has the smallest R-squared value and highest MSE and MAE value.

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



PSO+MSE



PSO+MAE

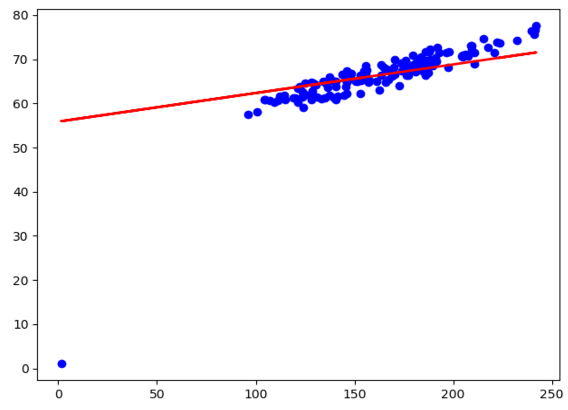
d) Compare the computational time of the BGD+MSE, MiniBatchBGD+MSE and PSO+MSE methods, find out the fastest one and slowest one, and explain why.

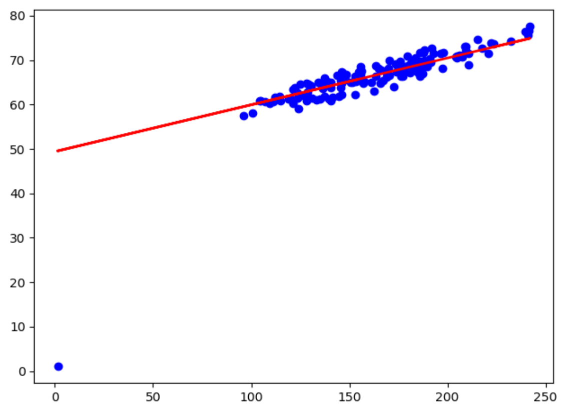
Metric Type	Optimizer Type	Execution Time
MSE	**BGD**	**0.002 seconds**
MSE	MiniBGD	0.008seconds
MSE	PSO	0.321 seconds
MAE	PSO	0.165 seconds

Reply: Looking at the table, **BGD+MSE** is the fastest, because the data set is not very large, and the array multiplication function in NumPy library is optimised. The processing as a whole array in BGD is faster than separating it into batches, calculating and saving for each batch. However, when calculating a much larger data set, the **BGD metho**d can be slower than **MiniBGD**, and even may not be fit into the memory of a computer.

There are more calculation and looping steps in **PSO algorithm**, according to its definition. Also, because the exponential calculation is slower than subtraction calculation, **PSO is slower** than the other 2 optimizers in this case, and **PSO+MSE** is **slower** than **PSO+MAE.**

2. **On the dataset with outliers, i.e. Part2Outliers.csv, split the data and run the PSO+MSE, and PSO+MAE methods in the same way as on the Part2.csv dataset in Question 1. Then:**
   a) **Generate the scatter plot with the regression line learnt by PSO+MSE and PSO+MAE and the data points in the test set.**





PSO+MSE **PSO+MAE**

   b) **Compare the above two plots with the two plots you draw in Question 1(c), and discuss which of the two methods (PSO+MSE or PSO+MAE) is less sensitive to outliers and explain why.**

Optimizer Type	Outliers	Loss
PSO+ MSE	without	1.90
PSO+ MAE	without	1.15
PSO+ MSE	with	4.85
PSO+ MAE	with	1.24

Reply: According to the results, when **without outliers**, the performance of the 2 methods are similar, in terms of both loss and the slope and intercept of the regression equation (**refer to the first 2 rows of the table below and Question 1(c) of the graphs**). But **with outliers**, the performance of the 2 methods, **MSE+PSO method** have the more increase of loss, and the slope have more reduce. So **PSO+MAE** is **less sensitive** than PSO+MSE.

   c) **Discuss whether we can use gradient descent or mini-batch gradient descent to optimise MAE? and explain why.**

Reply: We can use mini-batch gradient descent to optimise MAE, but this is **not a good method to** well optimized by derivative. Because Batch gradient descent refers to calculating the derivative from all training data before calculating an update. But using **MAE**, the cost function is **linear**, which means the adjustment are the same for each loop of converging. Even though a smaller or changing learning rate can be used in MAE, the converging may overshoot, and **not as precise as MSE** in the final stage of convergence.