VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

COMP309 Assignment 4
Performance Metrics and Optimisation

Han Han (300474699)

Supervisor(s): Xue Bing

Part 1: Performance Metrics in Regression

1. Exploratory Data Analysis

The task was done in below steps:

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. FMT is the format will be used in `strftime` and `strptime` methods to record the model execution time with the accuracy of 1 μs.

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

Step 2 - Initial Data Analysis: The first and last 5 rows of raw data in diamonds.csv were printed first. There are 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 on. By printing the shape of the DataFrame, 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 of the raw data:

```
       Unnamed: 0      carat      depth      table          x          y          z      price
count   53940.000  53940.000  53940.000  53940.000  53940.000  53940.000  53940.000  53940.000
mean    26970.500      0.797     61.749     57.457      5.731      5.734      3.538   3932.799
std     15571.281      0.474      1.432      2.234      1.121      1.142      0.705   3989.439
min         1.000      0.200     43.000     43.000      0.000      0.000      0.000    326.000
25%     13485.750      0.400     61.000     56.000      4.710      4.720      2.910    950.000
50%     26970.500      0.700     61.800     57.000      5.700      5.710      3.530   2401.000
75%     40455.250      1.040     62.500     59.000      6.540      6.540      4.040   5324.250
max     53940.000      5.010     79.000     95.000     10.740     58.900     31.800  18823.000
```

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 has to 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.

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:

Cut					
Categorical Raw Data	'Ideal'	'Premium'	'Very Good'	'Good'	'Fair'
After Quantifying	100	90	80	70	60

Colour							
Categorical Raw Data	'D'	'E'	'F'	'G'	'H'	'I'	'J'
After Quantifying	100	90	80	70	60	50	40

Clarity								
Categorical Raw Data	'IF'	'VVS1'	'VVS2'	'VS1'	'VS2'	'SI1'	'SI2'	'I1'
After Quantifying	100	90	80	70	60	50	40	30

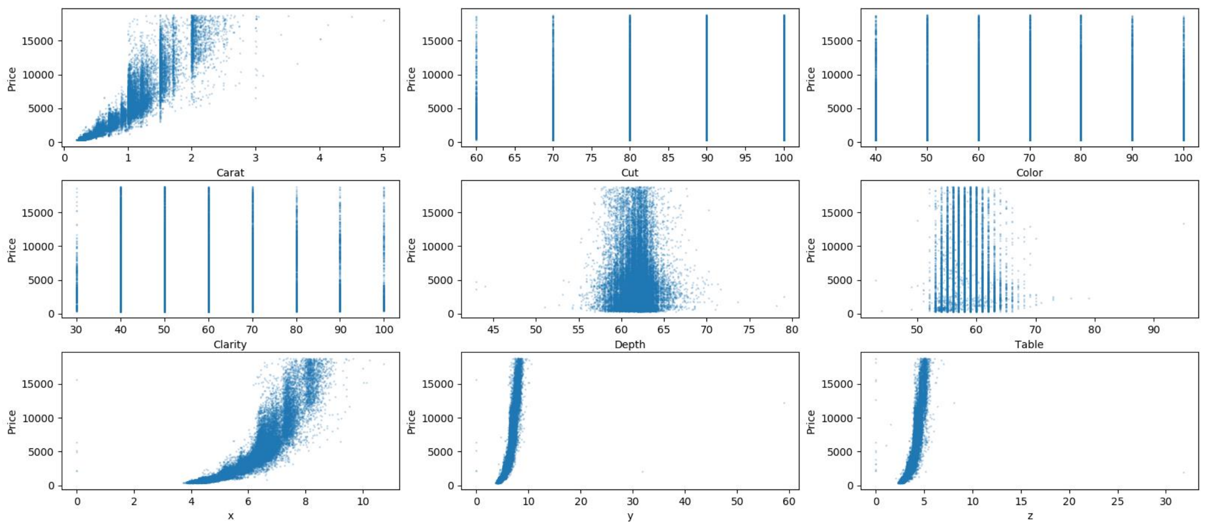
Step 3.3 - Data Pre-processing (Train-Test Split): The data set was split into 2 DataFrames and 2 Series, Xs_train_set, Xs_test_set, y_train_set, y_test_set, with the shape of (37758, 9), (16182, 9), (37758, 1) and (16182, 1), respectively. The `test_size` was set in the beginning of the program: 0.3.

Step 4 - Exploratory Data Analysis: The correlation between the features was explored. The result of correlation between the price and other columns is shown below:

```
         carat      cut    colour   clarity    depth     table       x        y        z     price
price    0.920   -0.051   -0.171    -0.144   -0.013     0.127    0.886    0.888    0.881     1.00
```

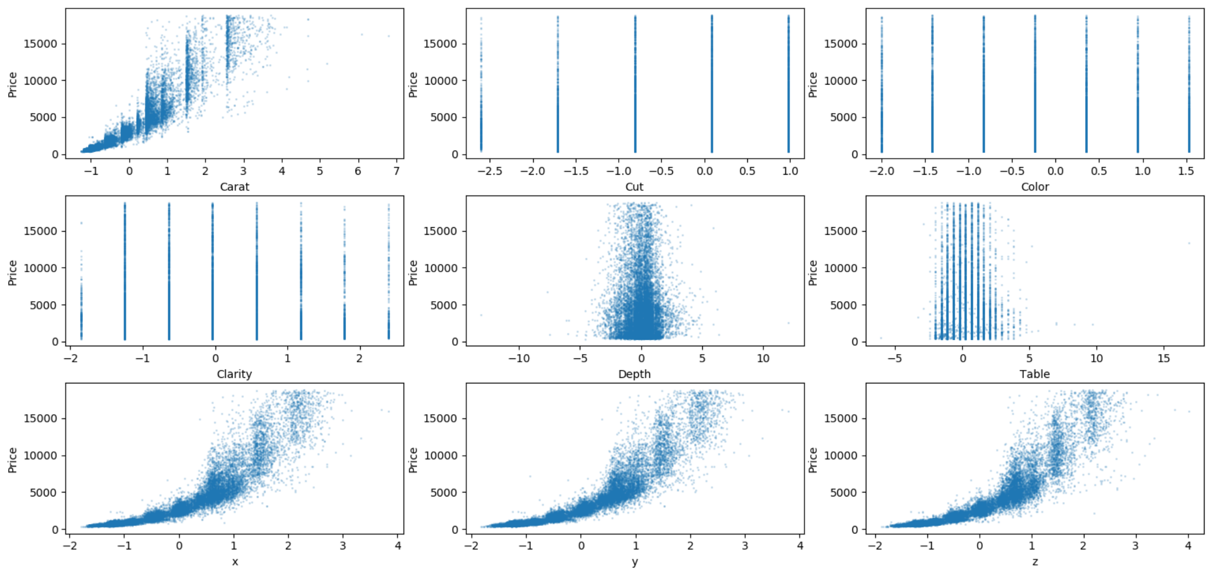
From the result, we can see the carat and size of the diamond have a strong correlation with the selling price.

The distributions of each feature are also plotted in scatter plots:



From the Price-y plot and the raw data, 2 outliers were found with y value of 58.9 and 31.8 (the 3rd highest y value is 10.54), 1 outlier was found with z value of 31.8 (the 2nd highest z value is 8.06). These 3 instances can be removed from dataset with those instances which have 0 value in x, y or z columns. After data cleansing, 23 instances were removed.

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 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: R^2, MSE, RMSE and MAE. The ranking in each column were added next to each one of them in italic font.

(Default Setting)	R^2	*R*	RMSE	*R*	MAE	*R*	MSE	*R*	Execution Time(s)	*R*
LinearRegression	0.91	*6*	1200.44	*6*	803.56	*6*	1441048.40	*6*	0.015624	*1*
KNeighborsRegressor	0.97	*3*	729.73	*3*	378.73	*4*	532509.66	*3*	0.541984	*6*
Ridge Regression	0.91	*6*	1200.51	*7*	803.63	*7*	1441231.18	*7*	0.031218	*2*
DecisionTreeRegressor	0.96	*4*	740.48	*4*	360.77	*3*	548317.03	*4*	0.156279	*5*
RandomForestRegressor	0.98	*1*	572.03	*1*	284.90	*1*	327212.81	*1*	0.993098	*7*
GradientBoostingRegressor	0.98	*1*	622.59	*2*	344.60	*2*	387621.92	*2*	0.996107	*8*
SGDRegressor	0.91	*6*	1209.19	*8*	811.65	*8*	1462142.32	*8*	0.041679	*3*
SVR	0.51	*10*	2854.23	*10*	1350.76	*10*	8146634.59	*10*	50.382575	*10*
LinearSVR	0.85	*9*	1561.06	*9*	850.37	*9*	2436903.76	*9*	0.046809	*4*
MLPRegressor	0.96	*4*	772.38	*5*	451.82	*5*	596567.12	*5*	20.935457	*9*

The top 3 algorithms in terms of R^2, 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 Linear Regression, Ridge Regression and SGD Regression all come from linear_model branch. Trees, Neighbors and Ensembles are slower, which take 5 times, 18 times and 30 times of the average time Linear models take. Multi-layer Perceptron regressor and Support Vector Regressor take the longest time, which are 700 times and 1700 times longer than the linear regression models.

2. Parameters Tuning

Another .py file was set up to iterate the important parameters settings in each algorithm to explore the best setting for each algorithm. Because there are random factors in these algorithms, even if running the same algorithm with the same parameters setting, there will be variance in the results.

Hence, in this report, the result, especially the R^2, keeps only 2 decimals, and only if the R^2 value increases by 0.01, the optimisations are regarded as effective.

The tunable parameters and parameters actually tuned in this experiment for each one of the 10 regression algorithms were summarised in below table. The best parameters combinations are given, and the R^2 values are compared before and after tuning.

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

After the parameter optimization, the R^2 results for Decision Tree Regression, Multi-layer Perceptron Regression, Linear SVR, and especially, SVR were improved significantly. The execution time of K Neighbors Regression, Ridge Regression, Random Forest Regression, SVR and MLP Regression were reduced.

(Optimized Setting)	R^2	*R*	RMSE	*R*	MAE	*R*	MSE	*R*	Execution Time(s)	*R*
LinearRegression	0.91	*6*	1200.44	*7*	803.56	*8*	1441048.40	*7*	0.031246	*2*
KNeighborsRegressor	0.97	*4*	656.54	*4*	329.54	*4*	431043.42	*4*	0.391984	*5*
Ridge Regression	0.91	*6*	1200.51	*8*	803.63	*9*	1441231.18	*8*	0.030781	*1*
DecisionTreeRegressor	0.97	*4*	748.82	*5*	363.03	*5*	560738.26	*5*	0.164174	*4*
RandomForestRegressor	0.98	*1*	561.13	*2*	280.67	*2*	314866.78	*2*	0.588720	*6*
GradientBoostingRegressor	0.98	*1*	523.57	*1*	272.65	*1*	274121.86	*1*	20.278558	*9*
SGDRegressor	0.91	*6*	1247.07	*9*	867.88	*10*	1555178.70	*9*	0.046894	*3*
SVR	0.89	*10*	1317.08	*10*	697.09	*6*	1734711.44	*10*	39.831736	*10*
LinearSVR	0.91	*6*	1200.11	*6*	800.75	*7*	1440263.90	*6*	2.744344	*7*
MLPRegressor	0.98	*1*	588.70	*3*	328.46	*3*	346562.45	*3*	16.417136	*8*

** R - Ranking*

If comparing the performance between these regression methods, Gradient Boosting Regression, Linear SVR and MLP Regression achieved higher ranking in R^2, MSE, MAE and RMSE. If talking about execution time, the improvement of R^2 of Linear SVR 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*
LinearRegression	*6*	*6*	*6*	*7*	*6*	*8*	*6*	*7*	*1*	*2*
KNeighborsRegressor	*3*	*4*	*3*	*4*	*4*	*4*	*3*	*4*	*6*	*5*
Ridge Regression	*6*	*6*	*7*	*8*	*7*	*9*	*7*	*8*	*2*	*1*
DecisionTreeRegressor	*4*	*4*	*4*	*5*	*3*	*5*	*4*	*5*	*5*	*4*
RandomForestRegressor	*1*	*1*	*1*	*2*	*1*	*2*	*1*	*2*	*7*	*6*
GradientBoostingRegressor	*1*	*1*	*2*	*1*	*2*	*1*	*2*	*1*	*8*	*9*
SGDRegressor	*6*	*6*	*8*	*9*	*8*	*10*	*8*	*9*	*3*	*3*
SVR	*10*	*10*	*10*	*10*	*10*	*6*	*10*	*10*	*10*	*10*
LinearSVR	*9*	*6*	*9*	*6*	*9*	*7*	*9*	*6*	*4*	*7*
MLPRegressor	*4*	*1*	*5*	*3*	*5*	*3*	*5*	*3*	*9*	*8*

* Green - Ranking up; Red - Ranking drops

Part 2: Performance Metrics in Classification

1. Exploratory Data Analysis & Data Pre-processing

In this classification task, adult.data and adult.test are given separately. First of all, the two files have to be read into memory as two Pandas DataFrame. Then, after printing the head of both DataFrames, there were no column label for each feature in the original data. After searching online, below column labels were added to both DataFrames:

```
    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"]
```

By checking with `tail()` method, there are 32560 rows of data in training set, 16280 rows of data in test sets. The column numbers are 15 for both of them by checking the shapes of them. And there is no missing data in both data sets. If printing the statistical information of both data sets, we can consider the minimum and maximum data for each numerical column as reasonable. However, when printing the unique value for each column, there were some problems found:

a) There are so many unique values in Final_Weight column.
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. (Source: https://www.census.gov/programs-surveys/sipp/methodology/weighting.html) Considering the scope of this analysis is to classify whether a person in certain conditions in certain features can earn higher than $50k or not, and the highly discrete values in Final_Weight column which may induce high noise for the classifying process, this column was deleted from both DataFrames.

b) There are misleading information in the data set between Relationship & Sex
3 rows were found with either Husband in Relationship but Female in Sex, or Wife in Relationship but Male in Sex. There must be some mistakes when the respondents filled in the questionnaire form. And because there are only 3 rows of misleading data out of 32k rows, we can choose either remove them or not.

c) There is 1 respondent in training set but no one in test set comes from Holland
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.

d) The Education column was found have same meaning as Highest_Grade
The Education column was removed because of the repeated information, also because the Highest_Grade column has quantified unique values. And as the highest grade become higher, it does mean the person has a better education background.

e) The separate Husband and Wife in Relation repeats the information in Sex

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

After that, the 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. Results and Ranking of the 10 classification algorithms

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*

** R - Ranking*

After optimisation for most of the algorithms, the F1 score and AUC values were improved.

(Optimized Setting)	Acc	*R*	Prec	*R*	Rec	*R*	F1	*R*	AUC	*R*
KNeighborsClassifier	0.84	*8*	0.69	*8*	0.56	*10*	0.62	*8*	0.74	*9*
GaussianNaiveBayes	0.52	*10*	0.32	*10*	0.95	*1*	0.48	*10*	0.67	*10*
SVMClassifier	0.85	*4*	0.73	*3*	0.60	*7*	0.66	*4*	0.77	*5*
DecisionTreeClassifier	0.82	*9*	0.63	*9*	0.61	*6*	0.62	*9*	0.75	*8*
RandomForestClassifier	0.85	*6*	0.71	*7*	0.61	*5*	0.66	*5*	0.77	*4*
AdaBoostClassifier	0.87	*2*	0.77	*1*	0.64	*3*	0.70	*2*	0.79	*2*
GradientBoostingClassifier	0.87	*1*	0.76	*2*	0.67	*2*	0.71	*1*	0.80	*1*
LinearDiscriminantAnalysis	0.85	*7*	0.71	*6*	0.58	*9*	0.64	*7*	0.75	*7*
MultilayerPerceptronClassifier	0.85	*3*	0.72	*5*	0.62	*4*	0.67	*3*	0.77	*3*
LogisticRegression	0.85	*5*	0.73	*4*	0.59	*8*	0.65	*6*	0.76	*6*

** R - Ranking*

In below tables, the absolute value and ranking for each algorithm and statistical variable are labelled in red colour if improved, green colour if got worse by trade-off.

Value	*ACC*		*Prec*		*Rec*		*F1*		*AUC*	
Optimization	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*
KNeighborsClassifier	0.82	0.84	0.64	0.69	0.57	0.56	0.60	0.62	0.74	0.74
GaussianNaiveBayes	0.52	0.52	0.32	0.32	0.95	0.95	0.48	0.48	0.67	0.67
SVMClassifier	0.85	0.85	0.74	0.73	0.58	0.60	0.65	0.66	0.76	0.77
DecisionTreeClassifier	0.82	0.82	0.62	0.63	0.60	0.61	0.61	0.62	0.74	0.75
RandomForestClassifier	0.84	0.85	0.69	0.71	0.58	0.61	0.63	0.66	0.75	0.77
AdaBoostClassifier	0.86	0.87	0.75	0.77	0.61	0.64	0.67	0.70	0.77	0.79
GradientBoostingClassifier	0.87	0.87	0.80	0.76	0.61	0.67	0.69	0.71	0.78	0.80

LinearDiscriminantAnalysis	0.85	0.85	0.71	0.71	0.58	0.58	0.64	0.64	0.75	0.75
MultilayerPerceptronClassifier	0.85	0.85	0.70	0.72	0.63	0.62	0.66	0.67	0.77	0.77
LogisticRegression	0.85	0.85	0.73	0.73	0.59	0.59	0.65	0.65	0.76	0.76

Ranking	*ACC*		*Prec*		*Rec*		*F1*		*AUC*	
Optimization	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*	*bef*	*aft*
KNeighborsClassifier	*8*	*8*	*8*	*8*	*10*	*10*	*9*	*8*	*9*	*9*
GaussianNaiveBayes	*10*	*10*	*10*	*10*	*1*	*1*	*10*	*10*	*10*	*10*
SVMClassifier	*3*	*4*	*3*	*3*	*7*	*7*	*5*	*4*	*5*	*5*
DecisionTreeClassifier	*9*	*9*	*9*	*9*	*5*	*6*	*8*	*9*	*8*	*8*
RandomForestClassifier	*7*	*6*	*7*	*7*	*8*	*5*	*7*	*5*	*7*	*4*
AdaBoostClassifier	*2*	*2*	*2*	*1*	*3*	*3*	*2*	*2*	*2*	*2*
GradientBoostingClassifier	*1*	*1*	*1*	*2*	*4*	*2*	*1*	*1*	*1*	*1*
LinearDiscriminantAnalysis	*6*	*7*	*5*	*6*	*9*	*9*	*6*	*7*	*6*	*7*
MultilayerPerceptronClassifier	*5*	*3*	*6*	*5*	*2*	*4*	*3*	*3*	*3*	*3*
LogisticRegression	*4*	*5*	*4*	*4*	*6*	*8*	*4*	*6*	*4*	*6*

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 not balanced. If we just arbitrarily classify all the instances to the class 0 (earn <=50K), we can still get a not very bad but meaningless 0.76 accuracy. If we use this model to classify the test set, the classifier maybe cannot classify even one instance as class 1 (earn >50K).

Comparing to accuracy, AUC seems better to evaluate the performance of a classification. The higher the AUC the better. And at the same time, 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, or when the precision is low, recall is high, F1-score may be similar. So, for some cases, it is better to look into detail in both precision and recall after evaluating F1-score.

The best-performance algorithm in terms of the performance metrics is Gradient Boosting Classifier; the second one is Adaptive Boosting. Both of the two algorithms belong to Boosting algorithm. Boosting algorithm combines some average-performed models to get a better-performed model.

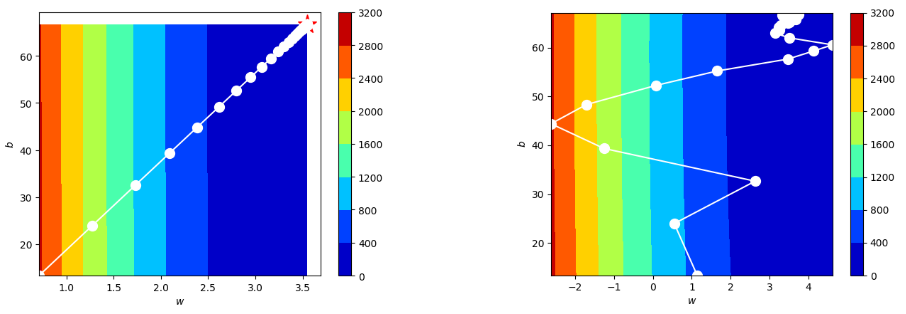
In Adaptive Boosting, several linear classifiers are used to classify the data one by one. After each sub-classification step, the weights of the wrongly classified data are increased. Finally, all these linear classifiers are combined together according to their weights. The weight of a certain sub linear classifier depends on its classification performance. And because the 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, BGD+MSE, MiniBatchBGD+MSE, PSO+MSE and PSO+MAE methods were implemented.

a) Gradient Descent paths of BGD+MSE and MiniBatchBGD+MSE



BGD uses all the training data in every loop of updating the weights to minimise the loss function. The loss moves toward the minimum directly since it considers all the data in the training set. The problem is the updating speed will be very slow if the dataset has large amount of training data.

SGD considers only one more new data point in the weights updating process. This can speed up the training very much. Also because of this, the noise cannot be filtered very well, which makes the path not move toward the minimum for every time. However, the general direction points toward the minimum. The high speed is the biggest advantage.

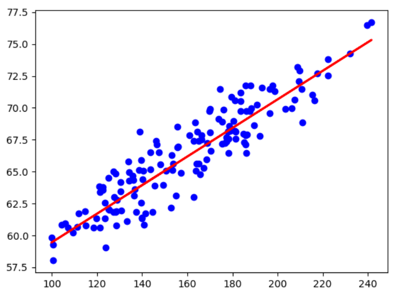
MiniBatchGD combines the advantages, but also compromises the shortages from both BGD and SGD. So, it's faster than BGD, and in the figure on the right-hand side, the path does not go toward the up-right corner directly.

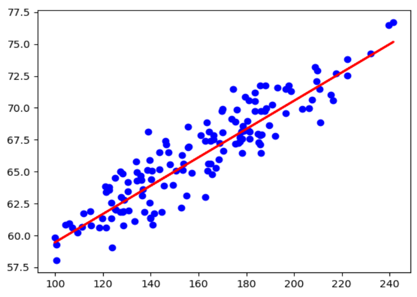
b) Performance metrics of the 4 models

Metric Type	Optimizer Type	R^2	MSE	MAE
MSE	MiniBGD	0.84	2.40	1.28
MSE	BGD	0.84	2.42	1.28
MSE	PSO	0.84	2.41	1.28
MAE	PSO	0.84	2.43	1.28

By comparing the performance metrics of the 4 models, the R^2 are almost the same if keeps only 2 decimals, which means the percentage of the interpretability of the fitting in these 4 models are almost the same. There is some difference in MSE for the 4 models, but the MAEs are almost the same, which means there are more outliers if using the MAE+PSO model to interpret than if using others. The MSE+MiniBGD has the minimum MSE performance, which means it can interpret the data with least outliers among the 4.

c) Scatter plots with regression line learnt by PSO+MSE and PSO+MAE in test set





d) Computation time comparison among the 4 models

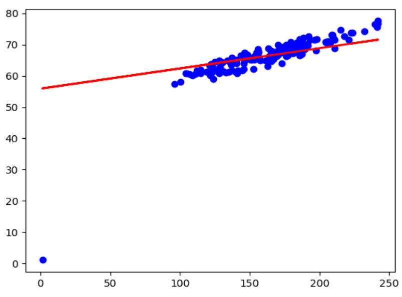
Metric Type	Optimizer Type	Execution Time
MSE	MiniBGD	0.027 seconds
MSE	BGD	0.002 seconds
MSE	PSO	0.279 seconds
MAE	PSO	0.152 seconds

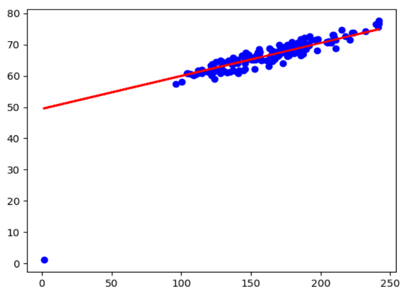
In this project, among the 4 models, BGD+MSE is the fastest, because the data set is not very large, and the array multiplication function in Numpy library is optimised. The processing as a whole array in BGD is faster than separating it into batches, calculating and saving for each batch. However, it is believed that, when calculating a much larger data set, the BGD method 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, PSO+MSE and PSO+MAE methods were implemented.

a) Scatter plots with regression line learnt by PSO+MSE and PSO+MAE in test set





b) Sensitivity of PSO+MES and PSE+MAE

When there is no outlier, 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). However, when there are outliers, the MSE is much more sensitive than if using MAE. The slope of the regression equation reduces from 0.11 to 0.06 because of the only 2 outlier points if the metric type is MSE. In contrast, the model using MAE is more stable. The reason is that MSE is more sensitive than MAE when there are outliers, especially when the outliers are far away from the main body, the square calculation in MSE is more sensitive than without the square calculation. The weight of outliers if using MSE is much larger than if using MAE.

Optimizer Type	Outliers	Metric Type	Loss	Regression Equation
PSO	without	MSE	1.87	Weight = 0.1119 * Height + 48.273
PSO	without	MAE	1.08	Weight = 0.1108 * Height + 48.388
PSO	with	MSE	4.85	Weight = 0.0647 * Height + 55.89
PSO	with	MAE	1.25	Weight = 0.1053 * Height + 49.425

c) Whether MAE can be used in GD or MiniBGD

The MSE cost function is parabolic if plotting vs. theta, which makes it can take smaller steps automatically when the converging process gets close to the optimised point (because the slope also becomes smaller). It means, even using a fixed learning rate, the learning rate can adjust by itself, and finally get a precise modelling result. However, if 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.