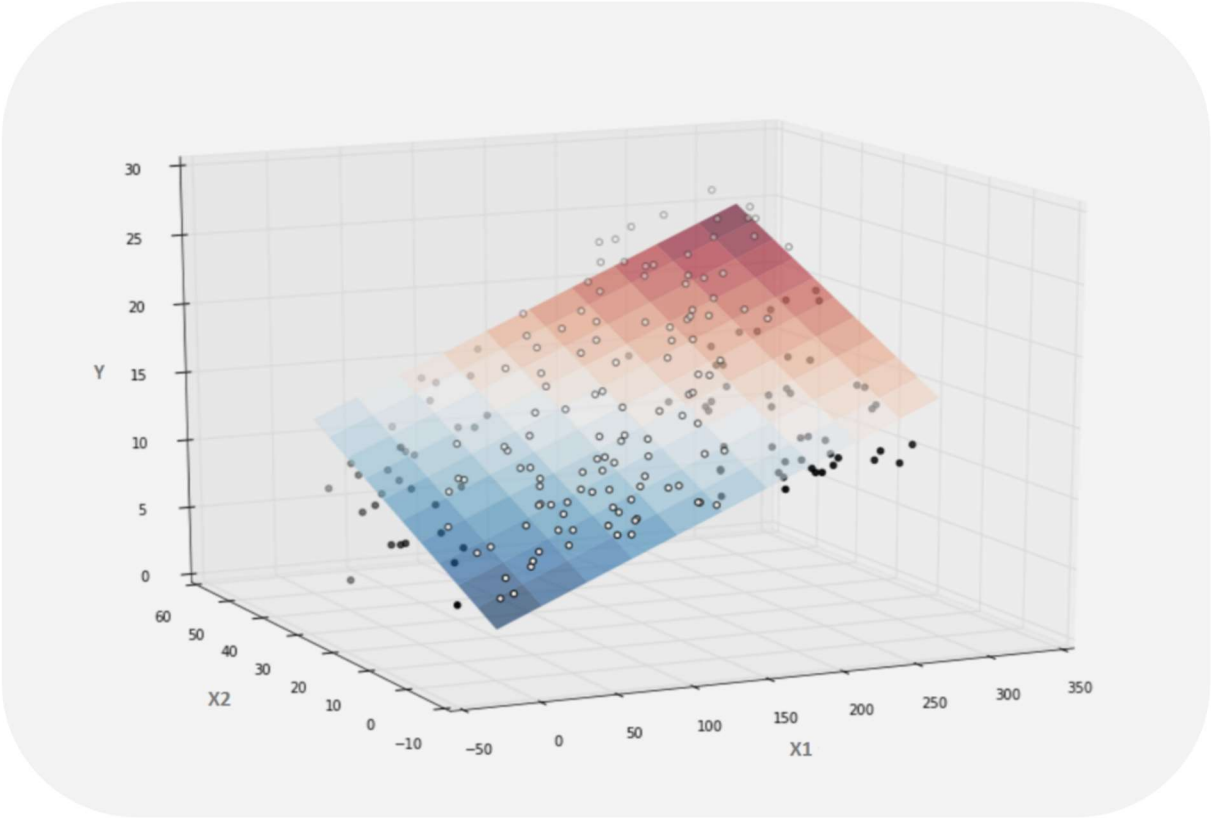
Regression Analysis and Results

On Single and Multiple Variable Datasets



Author: Andri Danusasmita

Email: andridsasmita@gmail.com

Website: https://github.com/adsasmita

Report Summary

This report contains full scope of regression analysis as well as the results from the predictive models. The two tasks dissected in the report can be classified as single variable and multiple variable regression problems. The first section of the report consists of data exploration, which seeks to explain and gain understanding of the two datasets given. The second section is an exposition of the methodology used in the analysis: what mathematical models to implement, which scientific codebase to use, and what differences between each approach. The third and last section is the curated results of each analysis, in the form of tables and visualizations.

As evident in the methodology section, it is found that both the solutions for problem 1 and problem 2 are linear in nature. This is supported by consistent accuracy results of 99% and greater (>0.99) for both training and validation datasets. While the solutions are arrived by different mathematical methods – analytical and numerical – the evaluation results are identical to 4 significant figures for all the algorithms explored. This bolsters the confidence in the final results concluded.

TABLE OF CONTENTS

1. Data Exploration

The first dataset contains 100 points, 10 independent variables and 1 target variable, while the second dataset consists of two files, with 1000 and 105 data points respectively. Both files in the second dataset contain 10 independent variables and 4 target variables.

1.1. Distribution Statistics

In problem 1 dataset, the 10 feature variables seems to be normally distributed at a mean of nearly zero, and standard deviation of about 1.0. The target variable also seems to be normally distributed – though shifted more to the positive side.

Statistics	1	2	3	4	5	6	7	8	9	10	Target
count	100	100	100	100	100	100	100	100	100	100	100
mean	-0.092	0.152	-0.101	-0.017	-0.232	-0.113	0.048	-0.033	0.031	-0.095	65.818
std	0.988	0.902	0.969	1.083	1.009	1.039	0.909	0.866	1.030	1.051	230.061
min	-2.659	-3.046	-2.773	-2.370	-2.583	-2.223	-2.256	-2.364	-2.835	-2.740	-580.287
0.25	-0.682	-0.359	-0.840	-0.742	-0.905	-0.881	-0.651	-0.682	-0.582	-0.793	-77.119
0.5	-0.202	0.042	-0.039	-0.082	-0.391	-0.194	0.101	-0.033	0.121	-0.153	73.780
0.75	0.416	0.707	0.598	0.690	0.444	0.432	0.777	0.593	0.626	0.561	222.253
max	2.696	2.381	1.930	2.594	2.011	2.759	2.304	2.117	2.412	2.257	696.232

Table 1. Distribution Statistics for Problem 1 Dataset

Problem 2, on the other hand, consists of two separate .csv files – which are already commissioned as training data and checking / testing data. The 10 feature variables seem to all be normally distributed.

Statistics	Value00	Value01	Value02	Value03	Value04	Value05	Value06	Value07	Value08	Value09	TGT0	TGT1	TGT2	TGT3
count	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
mean	-0.054	-0.047	0.021	-0.019	-0.038	-0.047	-0.034	-0.016	0.059	-0.009	87.472	91.665	94.518	90.353
std	0.975	0.990	0.983	1.017	1.000	1.009	0.991	1.006	0.955	0.947	169.218	143.648	168.817	196.361
min	-2.760	-2.840	-3.020	-2.810	-3.740	-3.070	-2.840	-3.120	-3.390	-2.990	-355.860	-368.270	-436.150	-564.110
25%	-0.730	-0.710	-0.622	-0.760	-0.680	-0.720	-0.712	-0.763	-0.553	-0.653	-27.268	-9.247	-16.053	-33.853
50%	-0.060	-0.080	-0.005	-0.020	-0.030	-0.030	0.000	-0.050	0.050	-0.010	86.165	92.915	86.595	91.370
75%	0.600	0.640	0.690	0.673	0.630	0.638	0.620	0.653	0.670	0.640	198.670	189.028	213.725	222.695
max	2.930	2.660	2.870	3.210	3.310	3.430	3.170	3.800	2.980	3.090	723.190	585.310	650.520	705.460

Table 2. Distribution Statistics for Problem 2 Training Dataset.

The four target variables for problem 2, however, have more spread in terms of their standard deviation as well as minimum and maximum values. The higher indexed target variable (TGT3) seems to have higher spread / standard deviation than the lower indexed target variable (TGT0).

Statistics	Value00	Value01	Value02	Value03	Value04	Value05	Value06	Value07	Value08	Value09	TGT0	TGT1	TGT2	TGT3
count	105	105	105	105	105	105	105	105	105	105	105	105	105	105
mean	0.053	-0.178	0.048	-0.104	-0.175	-0.250	0.104	0.091	-0.082	0.027	79.242	86.531	96.332	65.882
std	1.107	1.310	1.055	1.178	1.047	1.166	1.044	1.277	1.120	1.170	324.081	282.587	325.356	381.824
min	-2.450	-2.840	-2.580	-2.720	-2.330	-3.070	-1.880	-2.800	-3.390	-2.260	-355.860	-368.270	-436.150	-564.110
25%	-0.790	-1.270	-0.740	-0.880	-0.860	-0.890	-0.550	-0.910	-0.810	-0.970	-237.580	-187.530	-234.240	-292.170
50%	0.120	-0.400	0.170	-0.100	-0.330	-0.190	-0.070	0.030	0.050	0.000	-2.460	-43.690	-15.290	-50.440
75%	0.790	0.750	0.720	0.740	0.630	0.540	0.690	1.180	0.610	0.950	391.770	361.970	422.210	436.170
max	2.760	2.500	2.460	2.720	2.490	2.110	2.530	3.800	2.980	3.090	723.190	585.310	650.520	705.460

Table 3. Distribution Statistics for Problem 2 Testing Dataset

1.2. Training and Testing Sets

In finding predictive relationship from corpus of data, it is important to separate the dataset into training and testing sets. The training set will be used as learning material for the model, while the test set is reserved, only to be used to assess the strength and accuracy of the predictive model. This means all analytical and numerical calculations when building the model will be performed only on the training data. Since problem 2 already contains two pre-commissioned training and checking files, it is not necessary to further split the dataset.

Problem 1, on the other hand, only contains 1 file containing all 100 datapoints. It is therefore necessary to split the data into training and testing sets. This is achieved by randomly choosing 20 points out of the 100 points as testing data, which translates to 20% of total data. This data will be unseen by our model when it performs training calculations.

Implementing the shuffling and splitting the data can be done using Scikit-Learn machine learning library, as shown in the code snippet below:

```
# Split Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

features_train, features_test, targets_train, targets_test = train_test_split( \
                        features, targets, test_size=0.20, random_state=12)

print('Number of training datapoints:', features_train.shape[0])
print('Number of testing datapoints:', features_test.shape[0])
```

```
Number of training datapoints: 80
Number of testing datapoints: 20
```

Figure 1. Splitting Problem 1 Data Points into 80% Training Set and 20% Testing Set

Now problem 1 dataset consists of 80 training and 20 testing data points, while problem 2 dataset consists of 1000 training and 105 data points. Datasets of both problem 1 and 2 are now ready to be processed into machine learning models.

Allocations of training and testing data points in the form of bar charts are shown below.

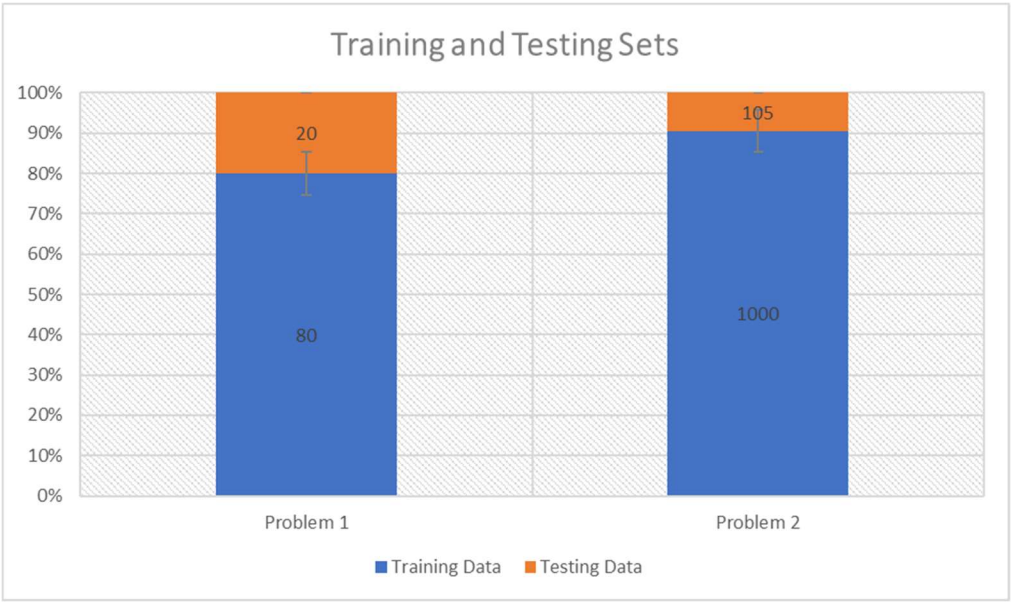


Figure 2. Allocation of Training and Testing Sets for Problem 1 and Problem 2

2. Methodology

This section provides an in-depth look into the methodologies used in developing our predictive models. First, the linearity of the regression needs to be clarified. Then, analytical as well as numerical method are performed to determine the unknown parameters of the regression functions.

2.1. Linearity vs. Non-Linearity

Since this is a regression task, there is prudence in determining whether the regression is linear in nature. A regression can be classified as linear when the target variables are related to the feature variables in a scalar multiplication relationship. Both problem 1 and problem 2 are multivariate regressions, since there are more than 1 feature variables that might have a relationship with the target variables. Illustrations of how the regression planes look like in linear and non-linear regressions are shown below.

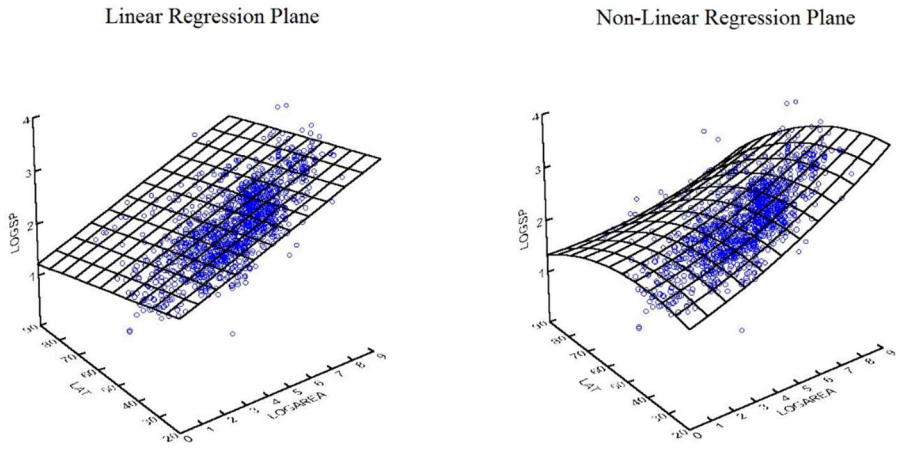


Figure 3. Example Illustrations for Linear and Non-Linear Regressions

Both algorithms for linear and non-linear regressions are performed in the analysis. It is discovered that for both problem 1 and problem 2 regressions, the resulting accuracies for linear regressions far outperform those for non-linear ones.

This report, therefore, focuses on finding solutions to the multivariate linear regression parameters for both problem 1 and problem 2. The solutions can be approached from two different angles: analytical and numerical (computational). The analytical solution involves parameterizing the dataset into matrices, then performing matrix multiplications on it. The numerical solution makes use of a simple neural network model which will be optimized using gradient descent method.

2.2. Analytical Approach: Ordinary Least Squares

Consider equation (1) which contains the expanded form of the multivariate linear equation. In simple matrix equation, equation (1) can be expressed in equation (2). The columns of X in (3) are each covariate for the target variable, with the first column being all 1's to include the bias value / zero-intercept in the model. Based on this model we get the matrix multiplication general equation as shown in (4). What we end up solving in matrix calculation, therefore, is contained in equation (5).

$$Y_1 = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12} + \ldots + \beta_p X_{1p} + \epsilon_1 \quad (1)$$

$$Y = X\beta + \epsilon \quad (2)$$

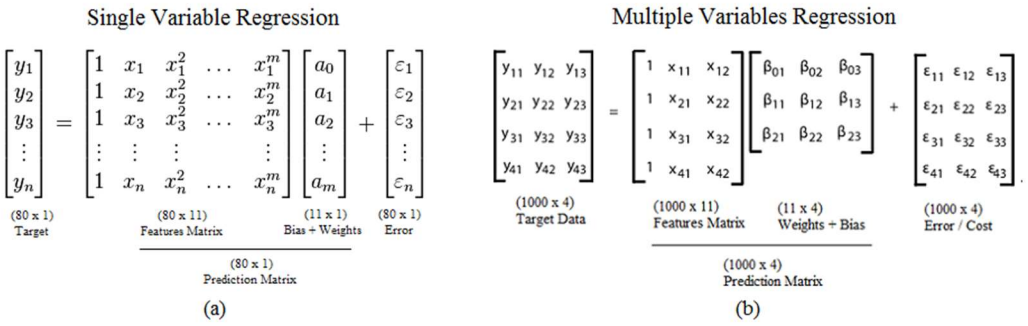
$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} \quad X = \begin{pmatrix} 1 & X_{11} & X_{12} & \ldots & X_{1p} \\ 1 & X_{21} & X_{22} & \ldots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & X_{n2} & \ldots & X_{np} \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix} \quad (3)$$

$$\hat{Y} = X\hat{\beta}. \quad (4)$$

$$\hat{\beta} = (X^T X)^{-1} X^T Y \quad (5)$$

Equation 1. Analytic General Equations for Linear Regression

The expanded matrix form of equation (3) are also illustrated in the examples below.



Equation 2. Matrix Representation of Datasets for (a) Problem 1 and (b) Problem 2

Solving for equation (5) can be done in Python using NumPy, as shown below. The function (developed in-house) will output the weights and bias parameters, as well as the predicted target values for both training and testing sets.

```
# Analytical Solution using Normal Equation
def analytic_normal(features_train, features_test, targets_train):

    # Append Bias Term to Features Matrix
    n_training_samples = features_train.shape[0]
    n_dimensions = features_train.shape[1]

    # Features + Bias Matrix
    X = np.reshape(np.c_[np.ones(n_training_samples),features_train],[n_training_samples,n_dimensions + 1])

    # Targets Matrix
    Y = np.reshape(targets_train,[n_training_samples,1])

    # Solve for Theta = (X(T).X)^(-1).X(T).Y
    # Theta will include Bias + Weights Matrix
    theta = np.dot(np.linalg.inv(np.matrix(np.dot(X.T, X))), np.dot(X.T, Y))
    theta = np.array(np.reshape(theta,(-1,1)))

    # Split Theta into Bias and Weights
    bias = theta[0]
    weights = theta[1:]
    weights = np.reshape(weights,(-1,))

    # Predictions on Training and Testing Sets
    pred_trn = np.dot(features_train, weights) + bias
    pred_tst = np.dot(features_test, weights) + bias

    return bias, weights, pred_trn, pred_tst
```

Figure 4. Implementation of Analytic Ordinary Least Squares Using NumPy Only

While solving ordinary least squares could be done in NumPy, it can also be done using Scikit-Learn linear regression module. The open source library could solve the multivariate linear regression parameters, as well as outputting the resulting R-squared accuracy value, as shown below.

```
# ML Linear Regression using Scikit-Learn
from sklearn.linear_model import LinearRegression

LinReg = LinearRegression()
LinReg.fit(features_train, targets_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Figure 5. Analytic Linear Regression Using Scikit-Learn Machine Learning Library

The analytic results from both in-house code and Scikit-Learn module are included in the results section. It is found that the resulting parameters and accuracy is identical for both analytical approaches.

2.3. Numerical Approach: Gradient Descent in Simple Neural Network

Numerical / computational models can be defined as performing iterative calculations (defined as training epochs) to the model, which if working correctly, will make the parameters converge to final values. The final values could be defined as optimized when the resulting error / cost function is the lowest globally.

A simple neural network model could be exploited to determine numerical solutions to the unknown parameters. It has to be noted that the neural network implemented only consisted of 1 output layer (no hidden layer), and it is fully-connected (dense). The input neurons will be placeholders to the number of feature variables that the data has. The output neurons will contain the predictions to the target variables. This means that for problem 1, there are 10 and 1 input and output neurons respectively. On the other hand, problem 2 neural network model will have 10 and 4 input and output neurons respectively.

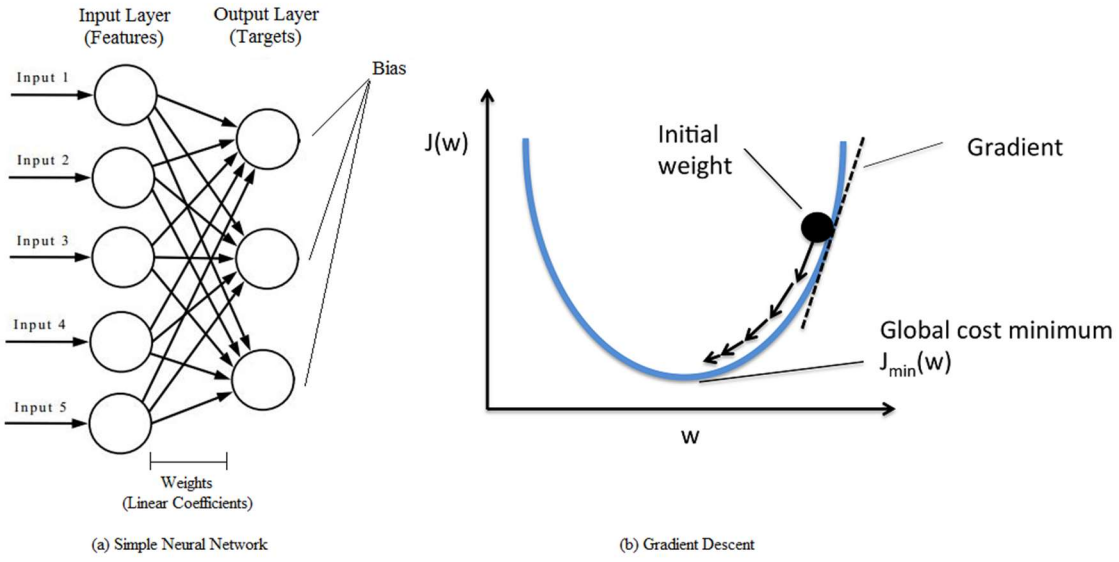


Figure 6. (a) Simple Neural Network Without Activation Function and (b) Illustration of Gradient Descent Algorithm

The output neurons will have bias values which will be added to the linear combination of the input neurons. This bias value is also an unknown parameter which will be solved numerically. The solutions that come from the neural network model will therefore in the form of neuron weights and biases.

It is also important to note that the output neurons will not have an activation functions in it. This is important because since this is a linear regression task, adding non-linear activation function will make the neural network unable to generalize the unknown linear parameters. This could result in finding the neural network unable to converge / converge very slowly.

The neural network will be optimized by gradient descent method. The loss function used in the model is mean squared error. The computational graphs of the neural network, the dense layer and the auxiliary nodes are shown below.

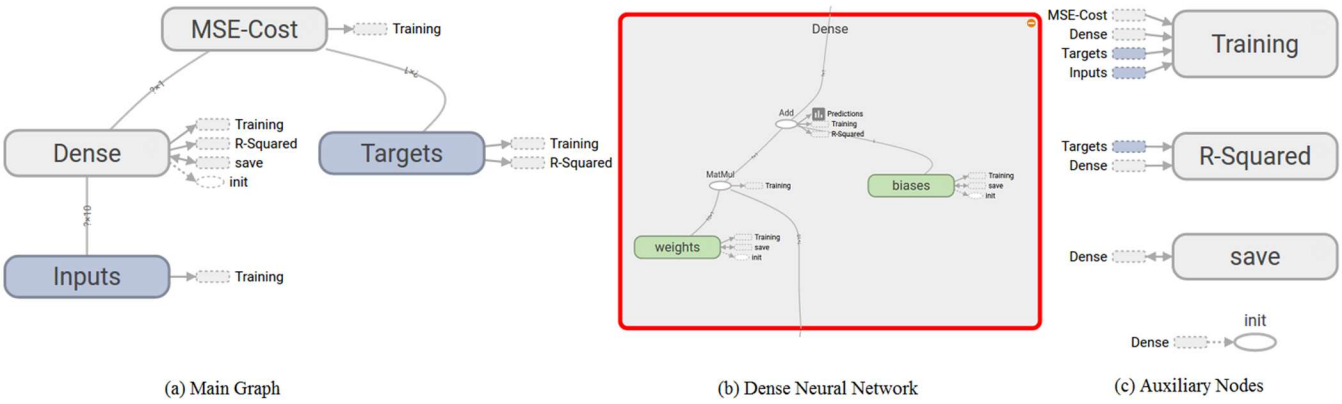


Figure 7. TensorBoard Illustrations of (a) Main Computation Graph (b) Expanded Dense Neural Layer and (c) Auxiliary Nodes

Building the neural network is done using an open-source deep learning library TensorFlow. The code implementation can be seen below. TensorFlow also has a built-in visualization tools for the computational graphs as well as training histograms, shown in this section and also in the appendices (Appendix A).

```
# Import TF Library
import tensorflow as tf
print("Using Google TensorFlow version", tf.__version__)
```

```
Using Google TensorFlow version 1.0.0
```

```
# Build NN Architecture
def neural_network(learning_rate, epochs, features_train, targets_train, \
                   features_test, targets_test, size_in, size_out):
```

(a)

(b)

Figure 8. (a) Building Neural Network and (b) Training Optimizer Computation Graph

3. Results

This section contains the curated results of each analysis, in the form of tables and visualizations. All the results shown in this section represents those that have ideal performance and far outclasses other algorithms explored. Both analytical solutions from in-house code and from Scikit-Learn module are identical, while the numerical solutions by TensorFlow neural network are almost identical (to 4 significant figures) to the analytical ones. All three solutions are curated since they produce consistent accuracy R-squared value of 99% and greater (>0.99) for both training and validation datasets.

3.1. Single Variable Regression

The linear regression solutions for problem 1 are shown in Table 4 below. Using any of the solution provided below would be optimal to predicting the model, though the accuracy in TensorFlow is slightly higher on testing data.

METHOD	Analytical		Neural Network
NAME	In-House	Scikit-Learn	TensorFlow
	Target	Target	Target
1	77.6802	77.6802	77.6799
2	34.1802	34.1802	34.1804
3	70.4355	70.4355	70.4357
4	1.9309	1.9309	1.9308
5	82.4040	82.4040	82.4038
6	97.1144	97.1144	97.1145
7	89.5215	89.5215	89.5213
8	60.8778	60.8778	60.8776
9	98.7051	98.7051	98.7049
10	3.0616	3.0616	3.0616
Bias	100.2805	100.2805	100.2802
R-Squared Training	0.99977110	0.99977110	0.99977110
R-Squared Testing	0.99974353	0.99974353	0.99974358

Table 4. Problem 1 Regression Results (Parameters & Accuracy)

To illustrate the high accuracy of the linear regression model, the scatterplots of training and testing data points in problem 1 are included below. It is evident that both scatterplots fall virtually on the regression lines.

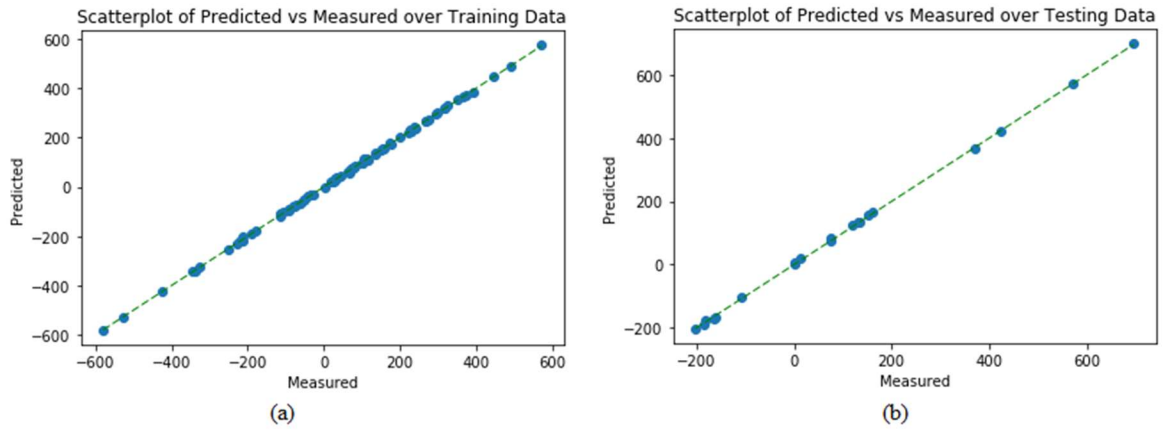


Figure 9. Regression Scatterplots of (a) Training Data and (b) Testing Data of Problem 1

3.2. Multiple Variables Regression

The linear regression solutions for problem 1 are shown in Table 4 below. Using any of the solution provided below would be optimal to predicting the model, though the accuracy in the analytical models slightly higher on testing data.

METHOD	Analytical								Neural Network			
NAME	In-House				Scikit-Learn				TensorFlow			
	TGT0	TGT1	TGT2	TGT3	TGT0	TGT1	TGT2	TGT3	TGT0	TGT1	TGT2	TGT3
Value00	20.905	41.553	50.182	82.005	20.905	41.553	50.182	82.005	20.905	41.552	50.182	82.004
Value01	88.158	45.450	62.035	69.434	88.158	45.450	62.035	69.434	88.157	45.449	62.034	69.433
Value02	13.218	43.906	69.673	63.396	13.218	43.906	69.673	63.396	13.218	43.906	69.672	63.395
Value03	9.644	40.141	52.256	73.243	9.644	40.141	52.256	73.243	9.644	40.142	52.257	73.244
Value04	8.246	33.614	5.062	14.807	8.246	33.614	5.062	14.807	8.246	33.614	5.062	14.807
Value05	75.145	36.346	5.585	81.583	75.145	36.346	5.585	81.583	75.144	36.345	5.585	81.582
Value06	77.771	50.227	31.071	13.284	77.771	50.227	31.071	13.284	77.770	50.226	31.071	13.284
Value07	65.366	46.032	98.183	67.699	65.366	46.032	98.183	67.699	65.366	46.033	98.182	67.700
Value08	13.211	35.045	53.477	79.024	13.211	35.045	53.477	79.024	13.211	35.044	53.477	79.023
Value09	74.469	82.574	49.184	43.228	74.469	82.574	49.184	43.228	74.468	82.573	49.184	43.227
Bias	100.104	100.019	100.096	99.802	100.104	100.019	100.096	99.802	100.103	100.018	100.095	99.801
R-Squared Training	0.99945035				0.99945035				0.99945035			
R-Squared Testing	0.99983871				0.99983871				0.99983868			

Table 5. Problem 2 Regression Results (Parameters & Accuracy)

This concludes the report. Further computational graphs of the neural network, as well as the training histograms are included in Appendix A section below. Other algorithms which were explored in the analysis is also included in Appendix B. They were not included in the main report because the resulting accuracies were either non-optimal or inconsistent between training and testing dataset – implying overfitting of the training data.

It is also worth noting that further insights into the results and methodologies can be obtained in the source code, which explores deeper into how each result and visualizations are arrived at. The source codes of this report can be found in the zipped solution of this report.

- End of Report -

Appendices

Appendix A: Neural Network Training Results

All illustrations provided in this section come solely by implementing TensorBoard visualization tool. It parses data from the namescopes defined in the neural network function, as evident in the source codes of this report.

The graphs below show how the Mean Squared Error (MSE) loss function decreases iteratively by each training epochs. The neural network trains in 2000 and 5000 training epochs for problem 1 and problem 2 respectively. The training times took about 1 and 3 seconds for problem 1 and problem 2 respectively.

For both problem 1 and problem 2, it is found that increasing training epochs (say, to 100,000) does not change the resulting solutions since the neural network has converged much earlier (about 1000 training epochs).

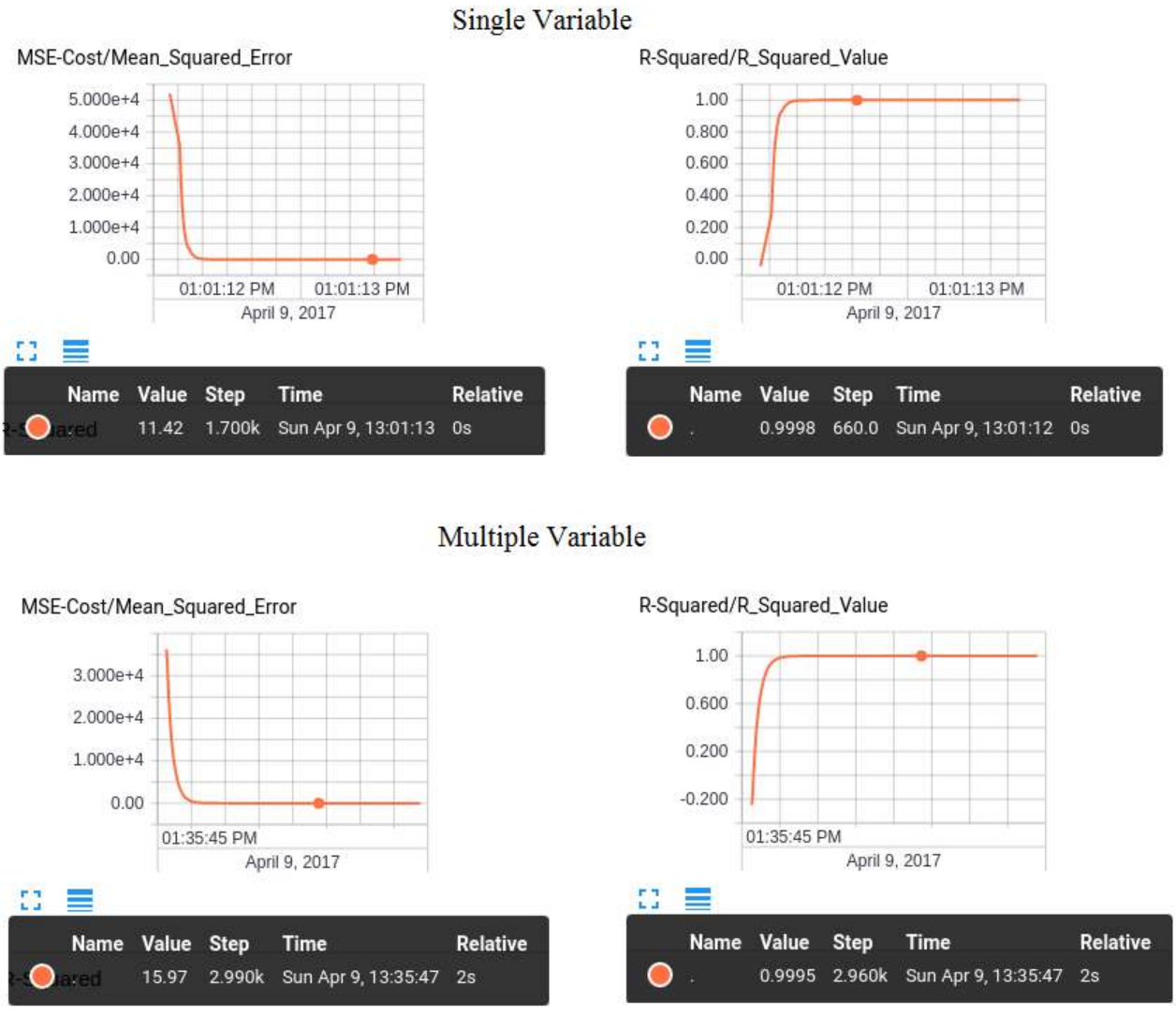


Figure 10. Training Loss Function and Accuracy Scalars Over Time

The illustrations below show the neural network training histogram and parameter distribution. It is plotted in three dimension, showing the cross-sectional view of the distributions plotted against time (offset view). The training histogram gives an intuition of how the biases, predictions and weights changes over training time, and how the values are distributed. The training distribution plot shows the range of the parameters changes over training time – as though seen from bird-eye view.

Figure 11. Training Histogram and Distribution of Neural Network in Problem 1

Figure 12. Training Histogram and Distribution of Neural Network in Problem 2

Appendix B: Other Algorithms Explored

Support Vector Machine Regressor (SVR)

```
# ML SVM Regressor
from sklearn.svm import SVR

svr = SVR(kernel='linear')
svr.fit(features_train, targets_train)

print("SVR R-Squared Score - Training Data:", svr.score(features_train,targets_train))
print("SVR R-Squared Score - Testing Data:", svr.score(features_test,targets_test))
```

```
SVR R-Squared Score - Training Data: 0.477028977496
SVR R-Squared Score - Testing Data: 0.463078164174
```

Figure 13. Support Vector Machine Regressor Algorithm and Its Resulting Accuracies

This algorithm implements Epsilon-Support Vector Regression. Since the resulting R-squared accuracies are not optimal, this implementation is not included in the main report.

Random Forest Ensemble Regressor

```
# ML Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor()
rfr.fit(features_train, targets_train)

print("Random Forest Regressor R-Squared Score on Training Data:", rfr.score(features_train,targets_train))
print("Random Forest Regressor R-Squared Score on Testing Data:", rfr.score(features_test,targets_test))
```

```
Random Forest Regressor R-Squared Score on Training Data: 0.910376793439
Random Forest Regressor R-Squared Score on Testing Data: 0.569385646525
```

Figure 14. Random Forest Ensemble Regressor Algorithm and Its Resulting Accuracies

This algorithm implements a random forest regressor. It is an ensemble learning estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging the improve the predictive accuracy and control over-fitting.

This algorithm produces high accuracy in the training data while having a much lower accuracy in the testing data, which suggests that the machine learning algorithm overfits the data and is unable to generalize to unseen testing data. Since the accuracies are inconsistent between the training and testing set, this implementation is not included in the main report.

Appendix C: Citations

- sklearn.model_selection.train_test_split (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)
- sklearn.linear_model.LinearRegression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
- sklearn.metrics.r2_score (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)
- tf.train.GradientDescentOptimizer (https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer)
- TensorBoard: Graph Visualization (https://www.tensorflow.org/get_started/graph_viz)
- tf.summary.histogram (https://www.tensorflow.org/api_docs/python/tf/summary/histogram)
- sklearn.svm.SVR (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html)
- sklearn.ensemble.RandomForestRegressor (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)