MSIS 5223: Tutorial 10 – Regression in Python

1. Assumption Validation

Prior to engaging in a regression analysis, you need to ensure your data meets five criteria. These criteria are assumptions and regression requires all of them to be valid:

- Linearity: linear relationship exists between *x* and *y* where $y = mx + b$
- Correlation: the independent variables (i.e. explanatory or predictor variables) are not related to each other
- Homoscedasticity: residuals exhibit a constant variance
- Independence: residuals are not related to each other
- Normality: residuals exhibit a normal distribution

The first assumption is fairly easy to check, but requires experience over time to make good judgment. Typically, a scatterplot or residual plot is used to assess linearity. A scatterplot is simply plotting the target variable on the *y*-axis and the predictor on the *x*-axis. Using the ozone.data.txt dataset, for example, to check the linearity of *Radiation* against *Ozone Concentration* you would type in the following and receive the scatterplot shown below:

```
In [4]: ozone_data.plot.scatter(x='rad', y='ozone')
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x14a89f
```

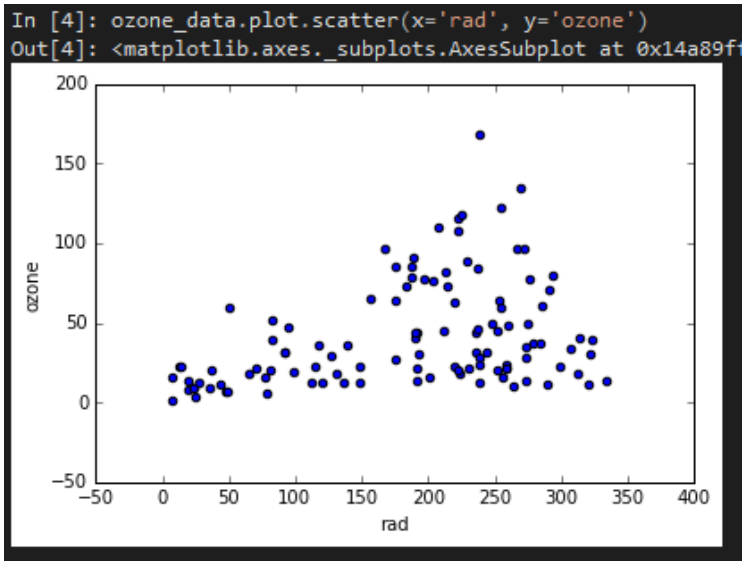

Figure 1 Scatterplot Code for Linearity Assessment

This is fairly linear and doesn't exhibit any curvature or other anomalies that would raise a red flag. You may be thinking that this data doesn't look very linear to you. This concern is sometimes raised because the data on the right-side of the plot has a

greater spread vertically than the data on the left-side. Linearity isn't concerned with the spread of the data. It merely is trying to fit a straight line to the data. This idea of spread is related to the third assumption, that of homoscedasticity. More on that later.

The next assumption to check in regression is that of multicollinearity. That is, your independent variables (i.e. predictor variables) are not related to each other. Your predictor variables should not influence each other. This type of relationship is alright between a predictor variable and your target variable (indeed, it should be highly related!). Two methods are typically used to assess collinearity (two related variables) or multicollinearity (three or more related variables).

The first method is correlation analysis. This is the more subjective assessment of the two methods. A correlation matrix typically provides the Pearson correlations among your variables. The code, along with the output, is presented below.

```
In [9]: ozone_data.corr()
Out[9]:
             rad      temp      wind     ozone
rad    1.000000  0.294088 -0.127366  0.348342
temp   0.294088  1.000000 -0.497146  0.698541
wind  -0.127366 -0.497146  1.000000 -0.612951
ozone  0.348342  0.698541 -0.612951  1.000000
```

Figure 2 Correlation Matrix for Ozone Data

The values range from -1.0 to 1.0 with an absolute value of 1.0 representing an exact relationship. Think of it as the extent to which the data points move together on a scatterplot. Looking at the names of the variables in the far left column, find ozone. Moving from left to right, the correlation values of each variable are listed. *Radiation* has a correlation of 34.8% with *Ozone Concentration*. This is a decent level of correlation and is good. *Temperature* is even better, at 69.9%

Returning to the assumption of multicollinearity, the correlation matrix should be used to look at the relationships among the independent variables. Looking at Figure 2, this would include the intersection of rad-temp, rad-wind, and temp-wind. Wind and temperature have a fairly high value of -49.7%. This isn't a cause for alarm. Typically, any correlation greater than the absolute value of 70.0% should be a cause for concern.

An important feature of the correlation matrix is the diagonal, where all the values are 100%. All of these values are at the intersection of a variable with itself. That is,

wind-wind, rad-rad-, temp-temp, and ozone-ozone. It shouldn't be surprising that a variable is highly correlated with itself.

This is important for the assumption. The closer a correlation score between two variables comes to 100%, the more likely they are the same variable. If you had done a principal components analysis with factor analysis, this problem with variables *x* and *y* wouldn't have occurred. Factor analysis would have shown that *x* and *y* are really the same thing.

The cutoff criteria of a correlation value of 70% is a rule-of-thumb. Some statisticians recommend a cutoff of 90%. Still, others say a lower value is more conservative. The correlation function in Pandas does not provide p-values. To obtain p-values, you can use a module from scipy. The code is shown in the figure below. Unfortunately, the code must be run for each correlation pair. The first value is the pearson correlation; the second is the p-value.

```
In [5]: from scipy.stats.stats import pearsonr
   ...: pearsonr(ozone_data.rad, ozone_data.ozone)
Out[5]: (0.34834169299360279, 0.00017931085716488428)

In [6]: pearsonr(ozone_data.wind, ozone_data.ozone)
Out[6]: (-0.61295075221446294, 8.6523964303648529e-13)
```

Figure 3 Correlations with P-Values

An alternative method to assessing multicollinearity is using a variance inflation factor, or VIF for short. Without getting too technical, a VIF is calculated for each variable after a regression is run and provides an index of the extent to which multicollinearity exists. The VIF scores are based on a regression model (shown later in section 3, Multiple Regression). Assuming you have already run a regression model, you can obtain the results shown in the figure below.

```
In [6]: linreg1.fit(ozone_data[['wind','temp']],ozone_data.rad)
   ...: vif1 = 1/(1 - linreg1.score(ozone_data[['wind','temp']],ozone_data.rad))

In [7]: vif1
Out[7]: 1.0952410154588483
```

Figure 4 VIF Score for Radiation

Unfortunately, in Python a function that automatically calculates the variance inflation factor does not come packaged with the regression modules. Each independent

variable requires a manual approach to calculation. The code below illustrates this by calculating the VIF for *Radiation*, *Wind*, and *Temperature*. Each independent variable takes a turn as the target variable with the other independent variables serving as the predictors. Note, the calculation is based off of the R-Square value. This is done using the module sklearn.linear_model and importing LinearRegression.

```
#Calculate VIF for Radiation
linreg1.fit(ozone_data[['wind','temp']],ozone_data.rad)
vif1 = 1/(1 - linreg1.score(ozone_data[['wind','temp']],ozone_data.rad))

#Calculate VIF for Wind
linreg1.fit(ozone_data[['rad','temp']],ozone_data.wind)
vif2 = 1/(1 - linreg1.score(ozone_data[['rad','temp']],ozone_data.wind))

#Calculate VIF for Temperature
linreg1.fit(ozone_data[['rad','wind']],ozone_data.temp)
vif3 = 1/(1 - linreg1.score(ozone_data[['rad','wind']],ozone_data.temp))
```

Figure 5 VIF Score Calculation in Python

A lower VIF score is desirable. By conservative standards, values that are greater than 5.0 are considered multicollinearity; a less conservative approach indicates values greater than 10.0 have multicollinearity. I go with the lower, restrictive number of 5.0 because it is less debatable. In the results above, all three independent variables have low enough values to rule out multicollinearity. The assumption of no multicollinearity is valid for this model and data.

The third assumption assessed for regression is homoscedasticity, or constant variance. That is, the spread of the data over *x* and *y* is consistent and doesn't change. When the data is not evenly distributed across the range of data, the data is said to have heteroscedasticity. Only a single plot is produced, plotting the residual values on the *y*-axis and the predicted values (sometimes called the fitted values) on the *x*-axis. Just like the VIF scores, this can only be produced after a regression model is built. To create this plot, use a different module, statsmodels.formula.api (see script file).

```
linreg2 = smf.ols('ozone ~ rad + wind + temp', ozone_data).fit()

#Assess homoscedasticity
plt.scatter(linreg2.fittedvalues, linreg2.resid)
plt.xlabel('Predicted/Fitted Values')
plt.ylabel('Residual Values')
plt.title('Assessing Homoscedasticity')
plt.plot([-40, 120],[0, 0], 'red', lw=2)   #Add horizontal line
plt.show()
```

Figure 6 Plotting Predicted Values vs. Residuals

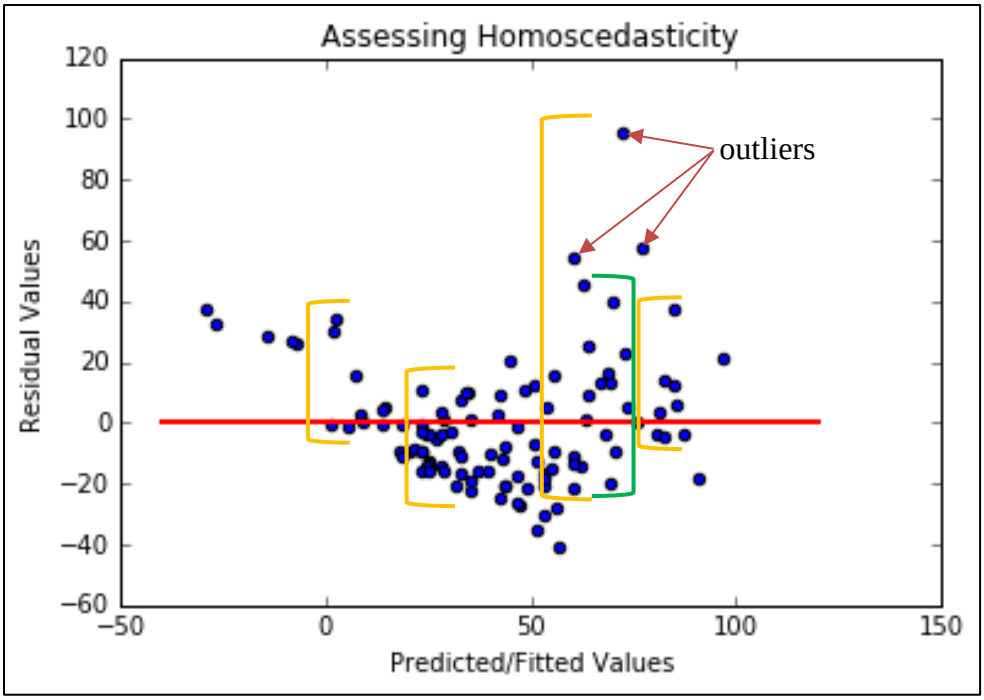

Figure 7 Checking for Homoscedasticity

The data points in Figure 11 are not evenly distributed across the *x*-axis. I have placed four orange brackets at key areas on the plot showing the spread of the data. The tail ends appear to have a fairly even spread, while a section in the middle has a large spread very different from the rest of the body. Notice that data points 23, 34, and 77 appear to be causing this large spread. The green bracket shows where the spread would be if those three data points were removed. It is possible that they are outliers. Their presence is causing heteroscedasticity, or non-constant variance. With the outliers present, this wouldn't pass the test for this assumption, and regression would be ruled out. It is still too early to decide what possible remedies might be taken.

The fourth assumption to check is that of independence of residuals. The Durbin-Watson test is a more objective assessment. Within Python, the statistic is obtained by viewing the summary results of the regression object. This output comes standard along with the other output. The result is not significant; therefore, there is no evidence to suggest this assumption is violated.

Omnibus:	38.276	Durbin-Watson:	1.935
Prob(Omnibus):	0.000	Jarque-Bera (JB):	84.526
Skew:	1.360	Prob(JB):	4.42e-19
Kurtosis:	6.297	Cond. No.	2.50e+03

Figure 8 Durbin-Watson Test for Independence

The last assumption is that of normally distributed residuals. A majority of people confuse this with having a normal distribution for each variable. These are two different concepts. In a previous In-Class Exercise, you learned to assess normality using a QQ Plot and the Shapiro-Wilk test. Those assessments were for checking normality of individual variables, not an entire set of variables. While it is good to have normally distributed variables, it is not a required condition to conduct regression. The residuals, on the other hand, must be normally distributed. This can be assessed by creating a QQ Plot.

To the right is the QQ Plot for the ozone dataset. This plot is read just like you were checking the normality for a variable. Notice the right-tail of the data extends upward, away from the normal line (i.e. the dashed line). Also notice that the data points 23, 34, and 77 are at the tail end. Again, it is quite possible that these data points are outliers. With the data points in the model, the assumption of normally distributed residuals fails for the ozone data. One solution is to remove these three data points; another solution is to transform the data, forcing it to be more normal.

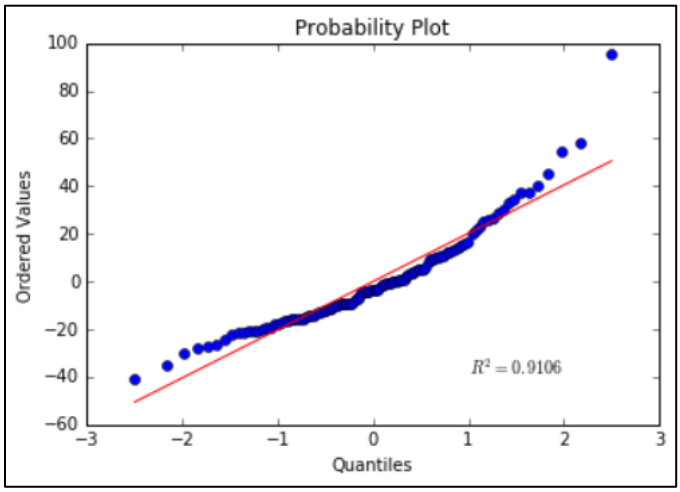

Figure 9 QQ Plot for Ozone Data

2. Linear & Multiple Regression

The two basic regression models are simple linear regression and multiple regression. These form the basis for understanding other types of regression such as polynomial regression and logistic regression. Recall that the equation for a line is

$$y = mx + b$$

where *y* is the dependent variable, *x* is the independent variable, and *m* and *b* are parameters, or constants. The parameter *b* is the *y*-intercept, or the value of *y* when *x* equals 0. The parameter *m* is the slope of the line and indicates to extent to which *y* changes with *x*.

Simple linear regression is merely the equation for a line. It contains only a single variable. The notation for the line equation changes when talking about regression. The equation is given as follows:

$$y = \beta_0 + \beta_1 x_1$$

Multiple regression takes this equation and extends it by introducing multiple slopes with multiple variables.

The equation is given as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where *n* is the number of variables if the model has more than two. The temptation exists to throw everything into a regression equation and see which variables work with the dependent variable. This is known as the kitchen sink model. Like with any model, the approach here should be about parsimony. That is, make it simple. The more complex the model becomes (i.e. the more variables in the model) the more difficult it becomes to implement it in the real world. Too many moving parts!

Return to the example of the ozone data. This dataset contains *ozone concentration*, *wind speed*, *air temperature*, and *intensity of solar radiation*. You must specify the regression model using notation similar to that of the

```
In [8]: linreg2 = smf.ols('ozone ~ rad + wind + temp', ozone_data).fit()
   ...: linreg2.summary()
Out[8]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  ozone   R-squared:                       0.606
Model:                            OLS   Adj. R-squared:                  0.595
Method:                 Least Squares   F-statistic:                     54.91
Date:                Tue, 13 Sep 2016   Prob (F-statistic):           1.45e-21
Time:                        14:46:23   Log-Likelihood:                -494.31
No. Observations:                 111   AIC:                             996.6
Df Residuals:                     107   BIC:                             1007.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept    -64.2321     23.042     -2.788      0.006      -109.910   -18.554
rad            0.0598      0.023      2.580      0.011         0.014     0.106
wind          -3.3376      0.654     -5.105      0.000        -4.634    -2.041
temp           1.6512      0.253      6.516      0.000         1.149     2.154
==============================================================================
```

Figure 10 Regression Output

regression equation. Using these variables, the regression equation would look like the following:

$$ozone = \beta_0 + \beta_1 rad + \beta_1 wind + \beta_2 temp$$

The output provides a lot of information about the regression model and can be overwhelming. The first place to start is at the bottom of the output. The first thing to check is the resulting F-statistic and p-value for the F-test. This is an ANOVA. Every p-value is associated with a hypothesis test. Each hypothesis test has a null hypothesis, designated as H_0, and at least one alternative hypothesis, designated as H_A. The F-test for a regression has the following hypotheses:

H_0: The null hypothesis states that all slopes, or βs, are equal to 0; that is, they have no effect on the dependent variable.

H_A: The alternative hypothesis states that at least one or more slopes is not equal to 0; that is, at least one slope has an effect on the dependent variable.

The null hypothesis is stating that all of the slopes of the line have no effect because 0 times any number is 0. Mathematically, it is expressed in the following way:

$$H_0: \beta_1 = \beta_2 = \cdots \beta_n = 0$$

If the p-value for this test is significant, then the null is rejected; at least one of the slopes influences the target variable. Note, however, that the F-test is unable to specify which variable is significant. It can only tell you that at least one of them is significant. *This is not a measure of how good the model is!* The coefficient of determination, or R-Squared, provides a measure of how good a regression model is. More on that later.

In the output above, the p-value for the F-statistic is less than $2.2e^{-16}$. Highly significant. If it had not been significant, then you would stop right away and not continue with the analysis. This means that either *radiation*, *wind*, or *temperature* is influential; or, possibly all three are influential.

With a significant F, how would someone know if a specific variable has an effect on the target variable? This is done by assessing the t-statistic associated with each independent variable. The hypotheses for the t-test in a regression are as follows:

H_0: The null hypothesis states that the slope, or β, is equal to 0; that is, this independent variable has no effect on the dependent variable.

H_A: The alternative hypothesis states that this slope is not equal to 0; that is, this independent variable has an effect on the dependent variable.

Mathematically, it is expressed as follows:

$$H_0: \beta_i = 0$$

Look at the p-value for each for each of the variables (the column on the far right of the middle highlighted area). *Radiation* has a p-value of 0.011, which is significant at the 0.05 level. This means it has a strong effect on *ozone*. The other two variables are even more significant, indicating they have a stronger effect on *ozone*. How strong of an effect do they have?

This information is easy to find. Looking at the regression output, the first column provides the coefficients, or estimates, of each variable. *Radiation* has a slope of 0.059, *wind* has a slope of -3.338, and *temperature* has a value of 1.651. Using this information, the regression equation becomes

$$ozone = -64.232 + 0.060rad - 3.338wind + 1.651temp$$

At this point in building the model, you can now check for homoscedasticity, variance inflation factor, and normally distributed residuals. As stated earlier in this exercise, this can only be done after the regression model is created.

Assume that no problems exist with this data. Two more steps are required for building the regression model. The model as a whole needs to be checked for how good it represents the data. Once done, the most important part of building a regression model is interpreting the results.

Looking at Figure 10 above, the Adjusted R-Squared value is 60%. This is a very good number for this initial model knowing the problems it has with two of the assumptions. Remember, use the Adjusted R-Squared value, not the R-Squared.

To interpret the model, simply use the coefficients, or slopes, to provide the change in the dependent variable wrought by the independent variable. For example, *wind* has a slope of -3.338. This means that for every unit increase in *wind*, *ozone* will decrease by -3.338. This indicates that in areas with higher wind speeds the level of ozone is depleted more. For *temperature*, every increase in 1 unit, *ozone* will increase by 1.651.

3. Logistic Regression

A logistic regression differs in many ways from linear regression. First of all, it is part of the generalized linear models, or GLM (pronounced as glim), as opposed to linear models (LM) like linear regression and polynomial regression. This means that the assumptions of regression do not apply to logistic regression. Secondly, the target variable is coded with a categorical value. If the categorical value is a binary value—typically 0/1, true/false, win/lose, male/female, -1/1—then it is binomial logistic regression. If the target variable has three or more categories, then it is referred to as multinomial logistic regression. Because binomial logistic regression is simpler to understand and more common, it is typically referred to as logistic regression. This tutorial will focus on binomial logistic regression.

The example data for this part of the tutorial is grading from a previous semester of a course I taught. The data includes only A and B grades. The dataset is class_performance.txt. The dataset contains seven columns of data. The dependent variable is *Grade*, which is a factor (i.e. categorical) containing A or B. In the logit model, the predictor variables will be *Project* and *Exam*, as well as an interaction term including the two predictors.

In Python, the first thing that needs to be done is convert the As and Bs to numerical values. The logit function within statsmodels module only accepts 0s and 1s as input for the target variable. The following figure shows how to perform this change.

```
In [6]: stdt_data.Grade.unique()
Out[6]: array(['A', 'B'], dtype=object)

In [7]: stdt_data.Grade.replace(['A','B'],[0,1], inplace=True)

In [8]: stdt_data.Grade.unique()
Out[8]: array([0, 1], dtype=int64)
```

Figure 11 Replacing Binary Values in Python

The original unique values were listed as “A” and “B” prior to the conversion; once changed, the unique values are switched over to “0” and “1.” A strong caution needs to be added here. If the datatype of the variable is categorical, you cannot simply change the values to something different. This is because the metadata of the categorical variable most likely doesn’t contain the new values you are using. Convert the variable to an object datatype, replace the values, then convert it back to a categorical.

```
In [9]: logreg1 = smf.logit('Grade ~ Project*Exam', stdt_data).fit()
Optimization terminated successfully.
         Current function value: 0.282878
         Iterations 11

In [10]: logreg1.summary()
Out[10]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Grade   No. Observations:                   69
Model:                          Logit   Df Residuals:                       65
Method:                           MLE   Df Model:                            3
Date:                Wed, 14 Sep 2016   Pseudo R-squ.:                  0.5774
Time:                        12:01:32   Log-Likelihood:                -19.519
converged:                       True   LL-Null:                       -46.184
                                        LLR p-value:                 1.559e-11
================================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
--------------------------------------------------------------------------------
Intercept     -389.5171    229.186     -1.700      0.089      -838.713    59.679
Project        421.3326    242.720      1.736      0.083       -54.389   897.054
Exam           794.9033    415.933      1.911      0.056       -20.311  1610.117
Project:Exam  -856.4395    440.265     -1.945      0.052     -1719.343     6.464
================================================================================
```

Figure 12 Logit Model of Grade Data

At first glance it would appear that the model is not very good. None of the predictors are very significant. The first inclination is to toss out the entire model and start from scratch. The other consideration may be to remove the variable *Project* because it has the worst p-value. In the tutorial for R, the deviance score was calculated for each variable. The two main effects were highly significant; the interaction term contributed the least amount of deviance. This would indicate the interaction term may be pulling away needed variance from the main effects, and should be removed from the model.

```
In [24]: logreg2 = smf.logit('Grade ~ Project + Exam', stdt_data).fit()
   ....: logreg2.summary()
Optimization terminated successfully.
         Current function value: 0.317922
         Iterations 8
Out[24]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                  Grade   No. Observations:                   69
Model:                          Logit   Df Residuals:                       66
Method:                           MLE   Df Model:                            2
Date:                Wed, 14 Sep 2016   Pseudo R-squ.:                  0.5250
Time:                        12:21:33   Log-Likelihood:                -21.937
converged:                       True   LL-Null:                       -46.184
                                        LLR p-value:                 2.949e-11
==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept      89.3345     23.781      3.757      0.000       42.725   135.944
Project       -83.3225     23.203     -3.591      0.000     -128.799   -37.846
Exam          -17.4205      5.683     -3.065      0.002      -28.559    -6.282
==============================================================================
```

Figure 13 Revised Logit Function

The results from the newly revised logistic function in Figure 13 agree with the previous conclusion. The interaction term was degrading the model and its removal results in better performance. The results indicate that both *Project* and *Exam* are influential in students obtaining a grade within the classroom.