Multiple Regression Model Evaluation

Introduction

Multiple regression models, like simple regression models, can be evaluated using R-Squared (coefficient of determination) for measuring the amount of explained variance, and the F-statistic for determining statistical significance. You also may want to consider using other metrics, including adjusted R-Squared and error-based metrics such as MAE and RMSE. Each metric provides slightly different information, so you will need additional information from stakeholders in order to choose the most appropriate one(s).

Objectives

You will be able to:

- Measure multiple linear regression model performance using adjusted R-Squared
- Measure regression model performance using error-based metrics

Multiple Regression Model with Auto-MPG Dataset

In the cell below we load the Auto MPG dataset and build a StatsModels multiple regression model that uses the `weight` and `model year` attributes to predict the `mpg` attribute.

In [1]:
```
import pandas as pd
import statsmodels.api as sm
```

In [2]:
```
data = pd.read_csv("auto-mpg.csv")
data
```

Out[2]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
...	...	...	...	...	...	...	...	...	...
387	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
389	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
390	28.0	4	120.0	79	2625	18.6	82	1	ford ranger
391	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

392 rows × 9 columns

In [3]:
```
y = data["mpg"]
X = data[["weight", "model year"]]
```

In [4]:
```
model = sm.OLS(y, sm.add_constant(X))
results = model.fit()
```

Reviewing F-Statistic and R-Squared

First, we can check the model's F-statistic and F-statistic p-value to see if the model was statistically significant overall:

In [5]:
```
results.fvalue, results.f_pvalue
```

Out[5]:
```
(819.4730484488967, 3.3321631451383913e-140)
```

Based on this very small p-value, we can say that the model is statistically significant.

We can also check the R-Squared (coefficient of determination):

In [6]:
```
results.rsquared
```

Out[6]:
```
0.8081803058793997
```

This means that our model is explaining about 81% of the variance in `mpg`.

Limitations of R-Squared

R-Squared is conventionally the "to-go" measurement for overall model performance, but it has some drawbacks.

First, as we add more predictors, R-Squared is only going to increase. More predictors allow the model an extra degree of freedom in trying to approximate a target. But this doesn't necessarily mean we have a better model, since we're likely to start seeing coefficients that are unstable and potentially not statistically significant. (We'll also see that these models might violate some of the *assumptions* of linear regression, which we'll describe in more detail later.)

To address this issue, consider using **adjusted R-Squared**. This version of R-Squared includes a correction for the number of predictors used.

Second, "proportion of variance explained" may not be the best way to describe your model's performance. It is very sensitive to the variance in the available data, where a very small variance in the predictors can lead to a small R-Squared and a very large variance in the predictors can lead to a large R-Squared, regardless of how well the model actually describes the true relationship between the variables. It can also be challenging to use when communicating with non-technical stakeholders who are less likely to have an understanding of "variance".

To address this issue, consider using an **error-based metric**. These measure the performance in terms of the errors (residuals), using various techniques to aggregate and summarize them.

Adjusted R-Squared

Adjusted R-Squared is still following the fundamental concept of "proportion of variance explained", except that the numerator and denominator are adjusted based on the degrees of freedom for the residuals

Recall that this is the formula for R-Squared:

$$R^2 = \frac{ESS}{TSS} = \frac{\sum_i (\hat{y_i} - \bar{y_i})^2}{\sum_i (y_i - \bar{y_i})^2}$$

and that we can rewrite this to be:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_i (y_i - \hat{y_i})^2}{\sum_i (y_i - \bar{y_i})^2}$$

Adjusted R-Squared, also written $\bar{R^2}$, takes the RSS value and divides it by the degrees of freedom of the residuals, and takes the TSS value and divides it by the degrees of freedom of the total.

$$\bar{R^2} = 1 - \frac{RSS/df_{resid}}{TSS/df_{total}} = 1 - \frac{(RSS)}{(TSS)}\frac{(df_{total})}{(df_{resid})}$$

We can calculate it ourselves like this:

In [7]:
```
y_pred = results.predict(sm.add_constant(X)) # use fitted model to generate predictions for y
rss = ((y_pred - y) ** 2).sum()
rss
```

Out[7]: 4568.952041558536

In [8]:
```
intercept_only_line = pd.Series([y.mean() for x in range(len(y))])
tss = ((y - intercept_only_line) ** 2).sum()
tss
```

Out[8]: 23818.99346938776

In [9]:
```
df_resid = len(y) - sm.add_constant(X).shape[1] # num observations - num parameters (including constant)
df_resid
```

Out[9]: 389

In [10]:
```
df_tot = len(y) - 1 # number observations - 1
df_tot
```

Out[10]: 391

In [11]:
```
1 - (rss / tss) * (df_tot / df_resid)
```

Out[11]: 0.8071940863723529

If we already have the value of R-Squared, we can also calculate it like this:

$$ \large{ \bar{R^2} = 1 - (1 - R^2) \frac{df_{total}}{df_{resid}} } $$

In [12]:
```
1 - (1 - results.rsquared) * (df_tot / df_resid)
```

Out[12]:
```
0.8071940863723529
```

...but actually we don't have to do either of those things, because **StatsModels will also calculate adjusted R-Squared for us**!

In [13]:
```
results.rsquared_adj
```

Out[13]:
```
0.8071940863723529
```

If you compare this to the non-adjusted version of R-Squared, you'll see that it is fairly similar.

In [14]:
```
results.rsquared
```

Out[14]:
```
0.8081803058793997
```

However we typically prefer using adjusted R-Squared if we're doing multiple regression modeling because adjusted R-Squared essentially only increases when the increase in variance explained is more than what we would expect to see due to chance.

Error-Based Metrics

Some issues with R-Squared can't be addressed with small tweaks like adjusted R-Squared. [This lecture that went viral on Reddit in 2015 (https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/10/lecture-10.pdf)](), for example, argues that R-Squared is "useless". We won't go that far, but there are some cases where an "error-based" metric would be more appropriate than R-Squared.

While R-Squared is a ***relative*** metric that compares the variance explained by the model to the variance explained by an intercept-only "baseline" model, most error-based metrics are ***absolute*** metrics that describe some form of average error.

(Technically R-Squared is also based on errors, but it's more indirect, so when we refer to "error-based" metrics we mean those that are more directly measuring average error.)

Why Not Just Sum the Residuals?

If we just find the sum of the residuals and divide it by the number of observations, we will get a very tiny value, like this:

In [15]:
```
results.resid.sum() / len(y)
```

Out[15]:
```
2.878423123436324e-14
```

This is because the residuals are a combination of positive and negative values.

In [16]:
```
results.resid
```

Out[16]:
```
0       2.573765
1       0.827227
2       2.122784
3       0.102888
4       1.209001
          ...
387    -2.249356
388    10.373474
389    -0.532233
390    -2.343648
391     1.286399
Length: 392, dtype: float64
```

When you add a positive residual value (where the model predicted a value that was too low) and a negative residual (where the model predicted a value that was too high), these "cancel each other out". But from our perspective, both of those were errors, and both should be counted. If we want this to happen, we need to transform them in some way.

Mean Absolute Error

Mean absolute error (MAE) calculates the **absolute value** of each error before adding it to the sum. We can just use the `.abs()` method on the residuals series to do this:

In [17]:
```
results.resid.abs()
```

Out[17]:
```
0       2.573765
1       0.827227
2       2.122784
3       0.102888
4       1.209001
          ...
387     2.249356
388    10.373474
389     0.532233
390     2.343648
391     1.286399
Length: 392, dtype: float64
```

In [18]:
```
mae = results.resid.abs().sum() / len(y)
mae
```

Out[18]:
```
2.608011858238716
```

MAE Interpretation

Unlike R-Squared, the units of MAE are in the units of y. Our target variable is `mpg`, so this result is in miles per gallon.

For this specific MAE value, it means that ***our model is off by about 2.6 miles per gallon in a given prediction***.

Is this a "good" MAE? In general, a lower MAE is better, and the smallest theoretically possible MAE is 0.

Your MAE will use the same units as the target, so a "good" MAE varies accordingly. For example, an MAE of 100 would be pretty bad if the target is the fuel economy of a car in MPG, but an MAE of 100 would be excellent if the target is the price of a house in USD.

Like with any other metric, you would need more context from stakeholders to be able to answer this question more precisely.

Root Mean Squared Error

Another popular error-based metric is root mean squared error. ***Root mean squared error (RMSE)*** calculates the **squared** value of each error, sums them, then takes the **square root** at the end.

We can use broadcasting with `**` (since $\sqrt{x}$ is the same as $x^{0.5}$) to calculate this:

In [19]:
```
results.resid ** 2
```

Out[19]:
```
0        6.624267
1        0.684305
2        4.506212
3        0.010586
4        1.461683
           ...
387      5.059602
388    107.608970
389      0.283272
390      5.492688
391      1.654822
Length: 392, dtype: float64
```

In [20]:
```
rmse = ((results.resid ** 2).sum() / len(y)) ** 0.5
rmse
```

Out[20]:
```
3.414013752452535
```

Root Mean Squared Error Interpretation

Even though the formula is different from MAE, the RMSE value is conventionally interpreted the same way.

For this specific RMSE value, it means that ***our model is off by about 3.4 miles per gallon in a given prediction***.

Comparing MAE and RMSE

Since both of these metrics express error in the same units, how do you know which one to choose?

The main question to consider is whether linearly increasing errors should be seen as ***linearly*** worse or ***exponentially*** worse. Consider the comparison of an error of 1 vs. an error of 10. Clearly an error of 10 is worse, but how much worse is it? If 10 is 10x worse than 1, that means MAE is a good metric. If 10 is 100x worse than 1, that means RMSE is a good metric.

There are also contexts related to *gradient descent* where RMSE is more mathematically appropriate, which become relevant when we move beyond models with "closed-form" solutions like ordinary least squares.

Calculating Metrics with Scikit-Learn

Rather than performing the math yourself to calculate MAE or RMSE, you can import functions from scikit-learn to make the task a bit easier.

In [21]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

`mean_absolute_error` (documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html)) is very straightforward to use for calculating MAE. You pass in the true y values along with the predicted $\hat{y}$ values, and it returns a score.

In [22]:
```
mean_absolute_error(y, y_pred)
```

Out[22]:
```
2.608011858238716
```

`mean_squared_error` (documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html)) is a bit more complicated to use for calculating RMSE. In addition to passing in y and $\hat{y}$, you need to specify `squared=False` to get RMSE. If you skip that third argument (or pass in `True` rather than `False`), you'll get MSE (mean squared error, without taking the square root at the end), which is useful in some machine learning contexts but challenging to describe to a stakeholder.

In [23]:
```
mean_squared_error(y, y_pred, squared=False)
```

Out[23]:
```
3.414013752452535
```

Summary

In this lesson you learned about some additional ways to evaluate regression model performance: adjusted R-Squared and error-based metrics. Adjusted R-Squared, like R-Squared, measures the percentage of variance explained, but it is adjusted to account for additional predictors being added. You can get the adjusted R-Squared value by using the `rsquared_adj` attribute of the StatsModels summary. Error-based metrics, such as MAE and RMSE, summarize the average errors made by the model, rather than the variance explained. You can calculate them using functions imported from scikit-learn.