Jupyter: Intro to Data Science - Lecture 5 Regression in Python

Grant Long
CUNY City College

NYC Tech-in-Residence Corps

Data Dive Week 5: Regression in Python

Note that this notebook borrows heavily from the online resources (https://github.com/cs109/2015lab4) for CS109 at Harvard University (http://cs109.github.io/2015/pages/videos.html).

This week we take a look at some basic statistical concepts, with a particular focus on regression models. As we covered in the lecture portion () of this week's class, linear regression is used to model and predict continuous outcomes. Time permitting, we'll also discuss logistic regression, which is used to model binary outcomes.

Though the DataCamp course covered for homework used the `numpy` package for linear regression, we'll also touch upon `statsmodels` and `scikit-learn` in today's exercise.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import statsmodels.api as sm
from sklearn.linear_model import LinearRegression

%matplotlib inline
```

Purpose of linear regression

Given a dataset X and Y, linear regression can be used to:

- Build a **predictive model** to predict future values of X_i without a Y value.
- Model the **strength of the relationship** between each dependent variable X_i and Y
  - Sometimes not all X_i will have a relationship with Y
  - Need to figure out which X_i contributes most information to determine Y

A brief recap

Linear Regression (http://en.wikipedia.org/wiki/Linear_regression) is a method to model the relationship between a set of independent variables X (also knowns as explanatory variables, features, predictors) and a dependent variable Y. This method assumes the relationship between each predictor X is linearly related to the dependent variable Y.

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where ϵ is considered as an unobservable random variable that adds noise to the linear relationship. This is the simplest form of linear regression (one variable), we'll call this the simple model.

- β_0 is the intercept of the linear model
- Multiple linear regression is when you have more than one independent variable
  - $X_1, X_2, X_3, \ldots$

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

StreetEasy Rentals Data Set

Data from this week's exercise comes from StreetEasy (www.streeteasy.com), an online platform for real estate listings in New York City. The sample provided covers 5,000 listings for homes for rent in Manhattan, Brooklyn, and Queens in June 2016 and provides several features of interest in modeling rents, including neighborhood, subway access, building amenities,

Note that this data is provided for instructional purposes only and is not intended to be representative of all listings on StreetEasy or all homes for rent in New York City. Any analysis conducted as part of this exercise does not reflect the opinion or endorsement of StreetEasy or any of its affiliates.

In [ ]:
```
se_df = pd.read_csv('https://grantmlong.com/data/streeteasy_rents_june20
16.csv')
```

In [ ]:
```
se_df.head()
```

Data Exploration

Summarize and Plot a Histogram for the Target Variable

Is there anything surprising or interesting about this data?

```
In [ ]:
```

Feature Exploration

If we wanted to try to create a model to price any given apartment, what variables might be the most important?

- How many variables are at our disposal?
- Which are binary? Categorical? Continuous?
- Which variable make most sense to use from an intuitive standpoint?
- Identify which variable has the highest correlation with

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Scatterplots

- Create a scatterplot of `size_sqft` , `bathrooms` , and `floor` .
- Describe the relationship you see? Is it positive or negative? Linear? Non-linear?

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Modeling

Single Variable Linear Regression with `size_sqft`

- Use `numpy` to fit a simple linear regression
  - Calculate the slope and intercept using the `polyfit` function
  - Print the slope and intercept. How would you interpret these two numbers?
  - Based on this data, how much would expect a 700 square foot apartment to cost?

```
In [ ]: beta, alpha = np.polyfit(XXXX, XXXX, 1)
        print('beta: %0.3f, alpha: %0.1f.' % (beta, alpha))
        print()
```

```
In [ ]:
```

```
In [ ]:
```

Visualize the relationship.

- Plot the fitted line along with the scatter plot.
- Is this line a good fit?

```
In [ ]:
```

```
In [ ]:
```

Calculate the predicted rent and residual for each observation.

- Create columns in the `se_df` dataframe for `rent_predicted` and `rent_residual`
- Does this appear to fall in line with the assumptions we've described?

```
In [ ]: se_df['rent_predicted'] = se_df['rent'] * XXXX + XXXX
        se_df['rent_residual'] = XXXX - XXXX
```

```
In [ ]:
```

```
In [ ]:
```

Using `statsmodels` for Single Variable Linear Regression

- Use `statsmodels` to fit a simple linear regression with `size_sqft`.
  - Output the regression results.
  - Describe how this output compares to our α and β from `numpy`.

In [ ]:
```
# Add a constant to our existing dataframe for modeling purposes
se_df = sm.add_constant(se_df)

est = sm.OLS(se_df['rent'],
             se_df[['const', 'size_sqft']]
            ).fit()

print(est.summary())
```

In [ ]:

Using `statsmodels` for Multiple Linear Regression

- Still using `statsmodels`, add some variables to our exisiting regression. Can you get a better prediction?
  - Add a one or two variables at a time. What happens to our R^2?
  - Which variables are most significant? How does this change as we add more predictors?
  - With regression with many predictors, create a histogram of the residuals. How does this compare to the single variable case?
    - *Note: use can access the residuals using the* `est.resid` *attribute of the regression results.*

In [ ]:
```
est = sm.OLS(se_df['rent'],
             se_df[['const', 'size_sqft']]
            ).fit()

print(est.summary())
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Using `sklearn` for Multiple Linear Regression

`sklearn` is among the most popular packages for machine learning, and it's one we'll be using throughout the rest of the semester. It's syntax and functionality is a little different, but it gives us a little more flexibility around accessing and using the output, and also plays nice with modeling options beyond linear regression.

In [ ]:
```
# This creates a LinearRegression object
lm = LinearRegression()
lm
```

What can you do with a LinearRegression object?

In [ ]:
```
# Look inside linear regression object
# LinearRegression.<tab>
```

Main functions	Description
`lm.fit()`	Fit a linear model
`lm.predit()`	Predict Y using the linear model with estimated coefficients
`lm.score()`	Returns the coefficient of determination (R^2). *A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model*

What output can you get?

In [ ]:
```
# Look inside lm object
# lm.<tab>
```

Output	Description
`lm.coef_`	Estimated coefficients
`lm.intercept_`	Estimated intercept

Fit a linear model

The `lm.fit()` function estimates the coefficients the linear regression using least squares.

```
In [ ]: # Use sensible subset of predictors to fit linear regression model
        dependent_vars = ['bedrooms', 'bathrooms', 'min_to_subway', 'floor',
                          'building_age_yrs', 'no_fee', 'has_roofdeck',
                          'has_washer_dryer', 'has_doorman', 'has_elevator',
                          'has_dishwasher', 'has_patio', 'has_gym']

        X = se_df[dependent_vars]

        lm.fit(X, se_df.rent)

        # notice fit_intercept=True and normalize=True
        # How would you change the model to not fit an intercept term?
```

Estimated intercept and coefficients

Let's look at the estimated coefficients from the linear model using `lm.intercept_` and `lm.coef_`.

After we have fit our linear regression model using the least squares method, we want to see what are the estimates of our coefficients β_0, β_1, ..., β_{13}:

$$\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_{13}$$

```
In [ ]: print('Estimated intercept coefficient:', lm.intercept_)
        print('Number of coefficients:', len(lm.coef_))
```

```
In [ ]: # The coefficients
        pd.DataFrame(lm.coef_, index=dependent_vars,  columns = ['Est. Coefficie
        nt'])
```

Predict Prices

We can calculate the predicted prices ($\hat{Y}_i$) using `lm.predict`.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots \hat{\beta}_{13} X_{13}$$

```
In [ ]: # first five predicted prices
        lm.predict(X)[0:5]
```

```
In [ ]: _ = plt.hist(lm.predict(X))
        _ = plt.title('Predicted Rents (fitted values): $\hat{Y}_i$')
        _ = plt.xlabel('Monthly Rent')
        _ = plt.ylabel('Frequency')
```

Let's plot the true prices compared to the predicted prices to see they disagree, we saw this exactly befor but this is how you access the predicted values in using `sklearn`.

In [ ]:
```
_ = plt.scatter(se_df['rent'], lm.predict(X))
_ = plt.xlabel("Rents: $Y_i$")
_ = plt.ylabel("Predicted rents: $\hat{Y}_i$")
_ = plt.title("Rents vs Predicted Rents: $Y_i$ vs $\hat{Y}_i$")
_ = plt.plot([0, 20000], [0, 20000], linewidth=4, color='red')
```

Residual sum of squares

Let's calculate the residual sum of squares

$$S = \sum_{i=1}^{N} r_i = \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_i))^2$$

In [ ]:
```
print('%0.2f' %  np.sum((se_df['rent'] - lm.predict(X)) ** 2))
```

In [ ]:
```

```

Bonus Round: Feature Engineering

Our original data set featured information on borough, submarket, neighborhood - all different ways of slices up the city in *geographic* terms.

- To what extent do you think models will return different results across different boroughs?
- How might you include some or all of these geographic areas in the model?

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```