Fitting and Evaluating Simple Linear Regression Models

Justin Post

Modeling Ideas

What is a (statistical) model?

- A mathematical representation of some phenomenon on which you've observed data
- Form of the model can vary greatly!

Modeling Ideas

What is a (statistical) model?

- A mathematical representation of some phenomenon on which you've observed data
- Form of the model can vary greatly!

Simple Linear Regression Model

- $$\text{response} = \text{intercept} + \text{slope*predictor} + \text{Error}$$

$$Y_i = \beta_0 + \beta_1 x_i + E_i$$

- May make assumptions about how errors are observed

NC STATE UNIVERSITY

Simple Linear Regression Model

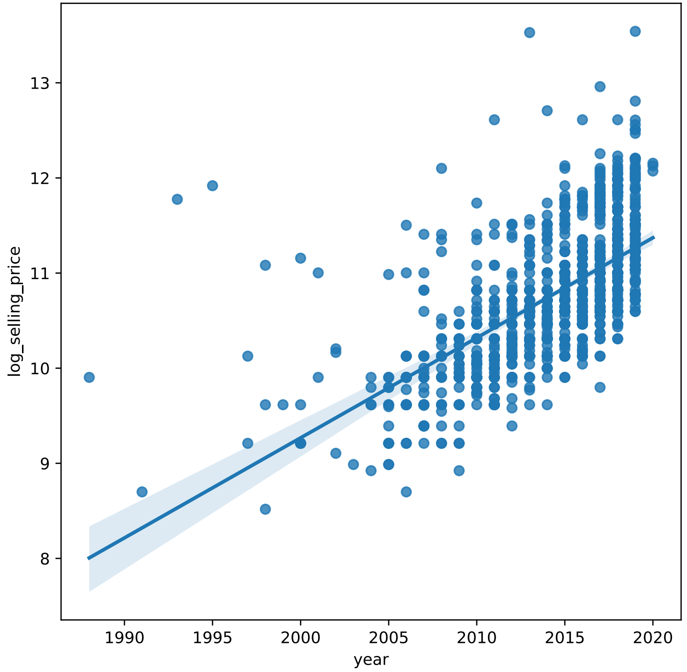
- First a visual

```
import pandas as pd
import numpy as np
import seaborn as sns
bike_data = pd.read_csv("https://www4.stat.ncsu.edu/~online/datasets/bikeDetails.csv")
bike_data['log_selling_price'] = np.log(bike_data['selling_price'])
bike_data['log_km_driven'] = np.log(bike_data['km_driven'])
```

Simple Linear Regression Model

- First a visual

```
sns.regplot(x = bike_data["year"], y = bike_data["log_selling_price"])
```



NC STATE UNIVERSITY

Statistical Learning

Statistical learning - Inference, prediction/classification, and pattern finding

- Supervised learning - a variable (or variables) represents an **output** or **response** of interest
    - May model response and
        - Make **inference** on the model parameters
        - **predict** a value or **classify** an observation

Goal: Understand what it means to be a good predictive model

NC STATE UNIVERSITY

Simple Linear Regression Model

Basic model for relating a numeric predictor to a numeric response

$$\text{response} = \text{intercept} + \text{slope*predictor} + \text{Error}$$

$$Y_i = \beta_0 + \beta_1 x_i + E_i$$

NC STATE UNIVERSITY

Simple Linear Regression Model

Basic model for relating a numeric predictor to a numeric response

$$\text{response} = \text{intercept} + \text{slope*predictor} + \text{Error}$$

$$Y_i = \beta_0 + \beta_1 x_i + E_i$$

Consider a data set on motorcycle sale prices

```
import pandas as pd
bike_data = pd.read_csv("https://www4.stat.ncsu.edu/~online/datasets/bikeDetails.csv")
print(bike_data.columns)
```

```
## Index(['name', 'selling_price', 'year', 'seller_type', 'owner', 'km_driven',
##        'ex_showroom_price'],
##       dtype='object')
```

```
bike_data.head()
```

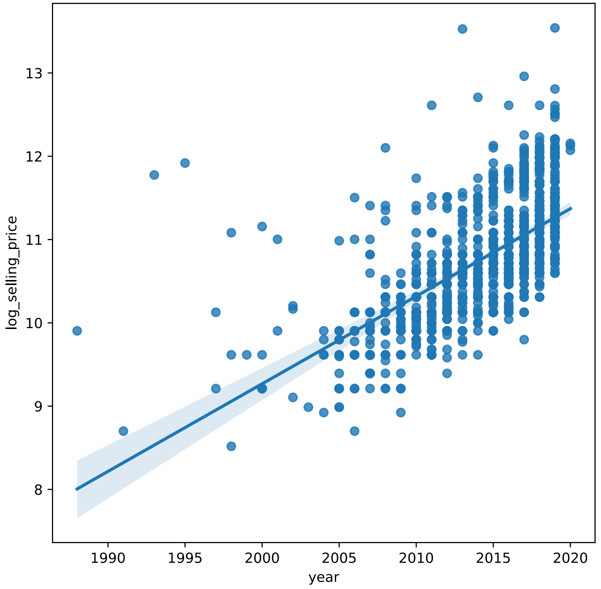
```
##                                     name  ...  ex_showroom_price
## 0                  Royal Enfield Classic 350  ...                NaN
## 1                                  Honda Dio  ...                NaN
## 2  Royal Enfield Classic Gunmetal Grey  ...           148114.0
## 3     Yamaha Fazer FI V 2.0 [2016-2018]  ...            89643.0
## 4                 Yamaha SZ [2013-2014]  ...                NaN
##
```

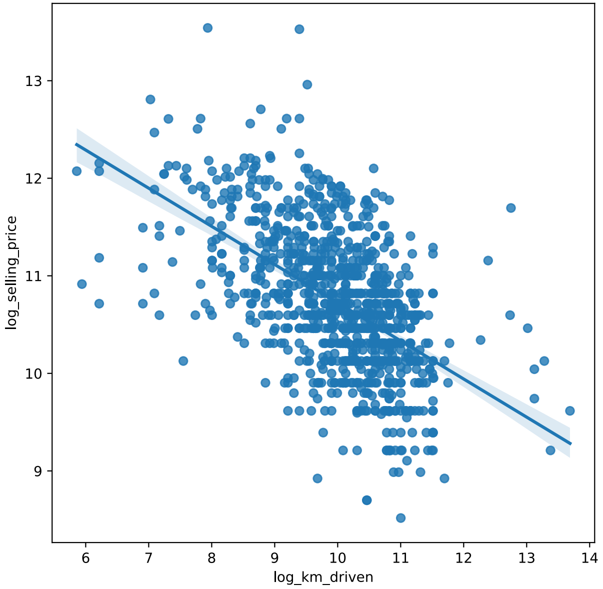
Find a 'Best' Fitting Line

- We define some criteria to **fit** (or train) the model

$$\text{Model 1: log\_selling\_price} = \text{intercept} + \text{slope*year} + \text{Error}$$

$$\text{Model 2: log\_selling\_price} = \text{intercept} + \text{slope*log\_km\_driven} + \text{Error}$$





Training a Model

- We define some criteria to **fit** (or train) the model
- **Loss function** - Criteria used to fit or train a model
    - For a given **numeric** response value, y_i and prediction, $\hat{y}_i$

$$y_i - \hat{y}_i, (y_i - \hat{y}_i)^2, |y_i - \hat{y}_i|$$

Training a Model

- We define some criteria to **fit** (or train) the model
- **Loss function** - Criteria used to fit or train a model
  - For a given **numeric** response value, y_i and prediction, $\hat{y}_i$

$$y_i - \hat{y}_i, (y_i - \hat{y}_i)^2, |y_i - \hat{y}_i|$$

- We try to optimize the loss over all the observations used for training

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Find a 'Best' Fitting Line

- In SLR, we often use squared error loss (least squares regression)
- Nice solutions for our estimates exist!

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Find a 'Best' Fitting Line

- In SLR, we often use squared error loss (least squares regression)
- Nice solutions for our estimates exist!

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
y = bike_data['log_selling_price']
x = bike_data['log_km_driven']
b1hat = sum((x-x.mean())*(y-y.mean()))/sum((x-x.mean())**2)
b0hat = y.mean()-x.mean()*b1hat
print(round(b0hat, 4), round(b1hat, 4))
```

```
## 14.6356 -0.3911
```

Find a 'Best' Fitting Line

- In SLR, we often use squared error loss (least squares regression)

- Nice solutions for our estimates exist!

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
y = bike_data['log_selling_price']
x = bike_data['log_km_driven']
b1hat = sum((x-x.mean())*(y-y.mean()))/sum((x-x.mean())**2)
b0hat = y.mean()-x.mean()*b1hat
print(round(b0hat, 4), round(b1hat, 4))
```

```
## 14.6356 -0.3911
```

- These give us the values to use with $\hat{y}$!

Simple Linear Regression Model in Python

- Can use `linear_model` from `sklearn` module to fit the model
- Note the requirements on the shape of `X` and the shape of `y` to pass to the `.fit()` method

```
print(bike_data['log_km_driven'].shape)
```

```
## (1061,)
```

```
print(bike_data['log_km_driven'].values.reshape(-1,1).shape)
```

```
## (1061, 1)
```

Simple Linear Regression Model in Python

- Can use `linear_model` from `sklearn` module to fit the model

```
from sklearn import linear_model
reg = linear_model.LinearRegression() #Create a reg object
reg.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'])
```

```
print(reg.intercept_, reg.coef_)
```

```
## 14.6355682846293 [-0.39108654]
```

Simple Linear Regression Model

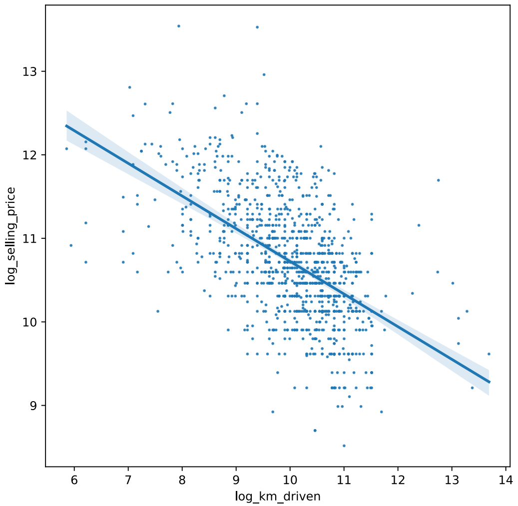
- Can use the line for prediction with the `.predict()` method!

```
print(reg.intercept_, reg.coef_)
```

```
## 14.6355682846293 [-0.39108654]
```

```
pred1 = reg.predict(np.array([[10], [12], [14]]))
pred1 #each of these represents a 'y-hat' for the given value of x
```

```
## array([10.72470291,  9.94252984,  9.16035677])
```



Recenter

Supervised Learning methods try to relate predictors to a response variable through a model

- Lots of common models
    - Regression models
    - Tree based methods
    - Naive Bayes
    - k Nearest Neighbors
- For a set of predictor values, each will produce some prediction we can call $\hat{y}$

Recenter

Supervised Learning methods try to relate predictors to a response variable through a model

- Lots of common models
    - Regression models
    - Tree based methods
    - Naive Bayes
    - k Nearest Neighbors
- For a set of predictor values, each will produce some prediction we can call $\hat{y}$

Goal: Understand what it means to be a good predictive model. **How do we evaluate the model?**

Quantifying How Well the Model Predicts

We use a **loss** function to fit the model. We use a **metric** to evaluate the model!

- Often use the same loss function for fitting and as the metric
- For a given **numeric** response value, y_i and prediction, $\hat{y}_i$

$$(y_i - \hat{y}_i)^2, |y_i - \hat{y}_i|$$

- Incorporate all points via

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Metric Function

- For a numeric response, we commonly use squared error loss as our metric to evaluate a prediction

$$L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

- Use Root Mean Square Error as a **metric** across all observations

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Commonly Used Metrics

For prediction (numeric response)

- Mean Squared Error (MSE) or Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE or MAD - deviation)

$$L(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

- Huber Loss
  - Doesn't penalize large mistakes as much as MSE

Commonly Used Metrics

For prediction (numeric response)

- Mean Squared Error (MSE) or Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE or MAD - deviation)

$$L(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

- Huber Loss
  - Doesn't penalize large mistakes as much as MSE

For classification (categorical response)

- Accuracy
- log-loss
- AUC
- F1 Score

Evaluating our SLR Model

- We could find our metric for our SLR model using the training data…
- Import our MSE metric from `sklearn.metrics`

```
import sklearn.metrics as metrics
pred = reg.predict(bike_data["log_km_driven"].values.reshape(-1,1))
print(np.sqrt(metrics.mean_squared_error(bike_data["log_selling_price"], pred)))
```

```
## 0.5947022682215317
```

```
print(metrics.mean_absolute_error(bike_data["log_selling_price"], pred))
```

```
## 0.46886132002881753
```

Useful for Comparison!

- Fit a competing model with `year` as the predictor

```
reg1 = linear_model.LinearRegression() #Create a reg object
reg1.fit(bike_data['year'].values.reshape(-1,1), bike_data['log_selling_price'])
```

```
<style>#sk-container-id-2 {color: black;background-color: white;}#sk-container-id-2 pre{padding: 0;}#sk-containe
```

```
print(reg1.intercept_, reg1.coef_)
```

```
## -201.06317651252067 [0.10516552]
```

- Compare the performance on the training data...

```
pred1 = reg1.predict(bike_data["year"].values.reshape(-1,1))
print(np.sqrt(metrics.mean_squared_error(bike_data["log_selling_price"], pred)),
      np.sqrt(metrics.mean_squared_error(bike_data["log_selling_price"], pred1)))
```

```
## 0.5947022682215317 0.548275146287923
```

Training vs Test Sets

Ideally we want our model to predict well for observations **it has yet to see**!

- For multiple linear regression models, our training MSE will always decrease as we add more variables to the model...

- We'll need an independent **test** set to predict on (more on this shortly!)

Recap

- SLR is one type of model for a continuous type response
- SLR Model is fit using some criteria (usually least squares, squared error loss)
- Must determine a method to judge the model's effectiveness (a metric)
  - Metric function measures *loss* for each prediction
  - Combined overall all observations
- To obtain a better understanding of the predictive power of a model, we need to apply our metric to prediction made on a different set of data than that used for training!