Prediction and Training/Test Set Ideas

Justin Post

Predictive Modeling Idea

- Choose form of model
- Fit model to data using some algorithm
    - Usually can be written as a problem where we minimize some loss function
- Evaluate the model using a metric
    - RMSE very common for a numeric response

Predictive Modeling Idea

- Choose form of model
- Fit model to data using some algorithm
    - Usually can be written as a problem where we minimize some loss function
- Evaluate the model using a metric
    - RMSE very common for a numeric response
- Ideally we want our model to predict well for observations **it has yet to see**!

NC STATE UNIVERSITY

Training vs Test Sets

- Evaluation of predictions over the observations used to *fit or train the model* is called the **training (set) error**
- Using RMSE as our metric:

$$\text{Training RMSE} = \sqrt{\frac{1}{\text{\# of obs used to fit model}} \sum_{\text{obs used to fit model}} (y - \hat{y})^2}$$

Training vs Test Sets

- Evaluation of predictions over the observations used to *fit or train the model* is called the **training (set) error**
- Using RMSE as our metric:

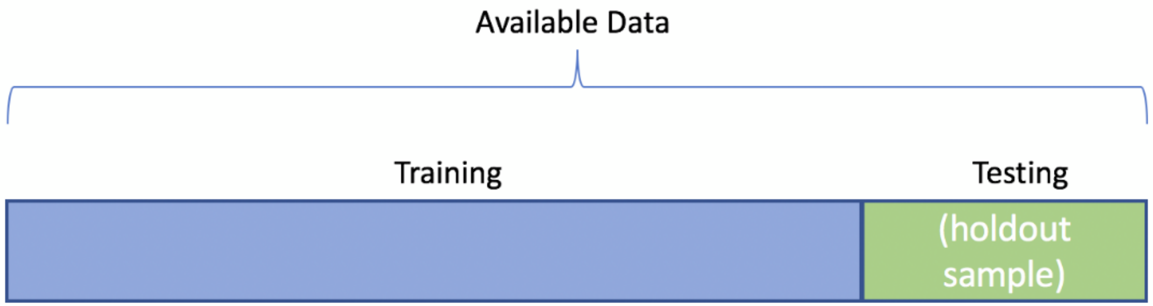
$$\text{Training RMSE} = \sqrt{\frac{1}{\text{\# of obs used to fit model}}\sum_{\text{obs used to fit model}}(y-\hat{y})^2}$$

- If we only consider this, we'll have no idea how the model will fare on data it hasn't seen!

Training vs Test Sets

One method is to split the data into a **training set** and **test set**

- On the training set we can fit (or train) our models
- We can then predict for the test set observations and judge effectiveness with our metric



Example of Fitting and Evaluating Models

Consider our data set on motorcycle sale prices

```
import pandas as pd
import numpy as np
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'])
print(bike_data.columns)
```

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

Example of Fitting and Evaluating Models

- Response variable of `log_selling_price = ln(selling_price)`
- Consider three linear regression models:

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

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

$$\text{Model 3: log\_selling\_price} = \text{intercept} + \text{slope*log\_km\_driven} + \text{slope*year} + \text{Error}$$

NC STATE UNIVERSITY

Fitting the Models with `sklearn`

```
from sklearn import linear_model
reg1 = linear_model.LinearRegression() #Create a reg object
reg2 = linear_model.LinearRegression() #Create a reg object
reg3 = linear_model.LinearRegression() #Create a reg object
reg1.fit(bike_data['year'].values.reshape(-1,1), bike_data['log_selling_price'])
reg2.fit(bike_data['log_km_driven'].values.reshape(-1,1), bike_data['log_selling_price'])
reg3.fit(bike_data[['year', 'log_km_driven']], bike_data['log_selling_price'])
```

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

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

```
print(reg2.intercept_, reg2.coef_)
```

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

```
print(reg3.intercept_, reg3.coef_)
```

```
## -148.79329107788155 [ 0.0803366  -0.22686129]
```

Example of Fitting and Evaluating Models

- Now we have the fitted models. Want to use them to predict the response

$$\text{Model 1: } \widehat{\text{log\_selling\_price}} = -201.06 + 0.105 * \text{year}$$

$$\text{Model 2: } \widehat{\text{log\_selling\_price}} = 14.64 - 0.391 * \text{log\_km\_driven}$$

$$\text{Model 3: } \widehat{\text{log\_selling\_price}} = -148.79 + 0.080 * \text{year} - 0.227 * \text{log\_km\_driven}$$

Example of Fitting and Evaluating Models

- Now we have the fitted models. Want to use them to predict the response

$$\text{Model 1: } \widehat{\text{log\_selling\_price}} = -201.06 + 0.105 * \text{year}$$

$$\text{Model 2: } \widehat{\text{log\_selling\_price}} = 14.64 - 0.391 * \text{log\_km\_driven}$$

$$\text{Model 3: } \widehat{\text{log\_selling\_price}} = -148.79 + 0.080 * \text{year} - 0.227 * \text{log\_km\_driven}$$

- Use the `.predict()` method

```
pred1 = reg1.predict(bike_data['year'].values.reshape(-1,1))
pred2 = reg2.predict(bike_data['log_km_driven'].values.reshape(-1,1))
pred3 = reg3.predict(bike_data[['year', 'log_km_driven']])
pd.DataFrame(zip(pred1, pred2, pred3, bike_data['log_selling_price']),
             columns = ["Model1", "Model2", "Model3", "Actual"])
```

```
##          Model1     Model2     Model3     Actual
## 0     11.266005  12.344609  12.077366  12.072541
## 1     11.055674  11.256811  11.285683  10.714418
## 2     11.160839  10.962225  11.195136  11.918391
## 3     10.845343  10.707789  10.806533  11.082143
## 4     10.424681  10.743366  10.505825   9.903488
```

Example of Fitting and Evaluating Models

- Find **training** RMSE

```
from sklearn.metrics import mean_squared_error
RMSE1 = np.sqrt(mean_squared_error(y_true = bike_data['log_selling_price'], y_pred = pred1))
RMSE2 = np.sqrt(mean_squared_error(bike_data['log_selling_price'], pred2))
RMSE3 = np.sqrt(mean_squared_error(bike_data['log_selling_price'], pred3))
print(round(RMSE1, 3), round(RMSE2, 3), round(RMSE3, 3))
```

```
## 0.548 0.595 0.511
```

- Estimate of RMSE is too **optimistic** compared to how the model would perform with new data! Redo with train/test split!

Train/Test Split

- `sklearn` has a function to make splitting data easy
- Commonly use 80/20 or 70/30 split

Train/Test Split

- `sklearn` has a function to make splitting data easy
- Commonly use 80/20 or 70/30 split

```
from sklearn.model_selection import train_test_split
#Function will return a list with four things:
#Test/train for predictors (X)
#Test/train for response (y)
X_train, X_test, y_train, y_test = train_test_split(
  bike_data[["year", "log_km_driven"]],
  bike_data["log_selling_price"],
  test_size=0.20,
  random_state=422)
```

Fit or Train Model

- We then fit the model on the training set

```
reg1 = linear_model.LinearRegression() #Create a reg object
reg2 = linear_model.LinearRegression() #Create a reg object
reg3 = linear_model.LinearRegression() #Create a reg object
reg1.fit(X_train['year'].values.reshape(-1,1), y_train.values)
reg2.fit(X_train['log_km_driven'].values.reshape(-1,1), y_train.values)
reg3.fit(X_train[['year', 'log_km_driven']], y_train.values)
```

Fit or Train Model

- We then fit the model on the training set

```
reg1 = linear_model.LinearRegression() #Create a reg object
reg2 = linear_model.LinearRegression() #Create a reg object
reg3 = linear_model.LinearRegression() #Create a reg object
reg1.fit(X_train['year'].values.reshape(-1,1), y_train.values)
reg2.fit(X_train['log_km_driven'].values.reshape(-1,1), y_train.values)
reg3.fit(X_train[['year', 'log_km_driven']], y_train.values)
```

- Can look at training RMSE if we want

```
train_RMSE1 = np.sqrt(mean_squared_error(y_train.values,
                                         reg1.predict(X_train['year'].values.reshape(-1,1))))
train_RMSE2 = np.sqrt(mean_squared_error(y_train.values,
                                         reg2.predict(X_train['log_km_driven'].values.reshape(-1,1))))
train_RMSE3 = np.sqrt(mean_squared_error(y_train.values,
                                         reg3.predict(X_train[['year', 'log_km_driven']])))
print(round(train_RMSE1, 3), round(train_RMSE2, 3), round(train_RMSE3, 3))
```

```
## 0.557 0.593 0.516
```

Test Error

- Now we look at predictions on the test set
  - Test data **not** used when training model

```
test_RMSE1 = np.sqrt(mean_squared_error(y_test.values,
                                        reg1.predict(X_test['year'].values.reshape(-1,1))))
test_RMSE2 = np.sqrt(mean_squared_error(y_test.values,
                                        reg2.predict(X_test['log_km_driven'].values.reshape(-1,1))))
test_RMSE3 = np.sqrt(mean_squared_error(y_test.values,
                                        reg3.predict(X_test[['year', 'log_km_driven']])))
print(round(test_RMSE1, 3), round(test_RMSE2, 3), round(test_RMSE3, 3))
```

```
## 0.513 0.603 0.491
```

- When choosing a model, if the RMSE values were 'close', we'd want to consider the interpretability of the model (and perhaps the assumptions if we wanted to do inference too!)

Recap

- Choose form of model
- Fit model to data using some algorithm
  - Usually can be written as a problem where we minimize some loss function
- Evaluate the model using a metric
  - RMSE very common for a numeric response
- Ideally we want our model to predict well for observations **it has yet to see**!