Bias-Variance Tradeoff - Lab

Introduction

In this lab, you'll practice the concepts you learned in the last lesson, bias-variance tradeoff.

Objectives

In this lab you will:

- Demonstrate the tradeoff between bias and variance by way of fitting a machine learning model

Let's get started!

In this lab, you'll try to predict some movie revenues based on certain factors, such as ratings and movie year. Start by running the following cell which imports all the necessary functions and the dataset:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import *
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_excel('movie_data_detailed_with_ols.xlsx')
df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Unnamed: 0	budget	domgross	title	Response_Json	Year	imd
0	0	13000000	25682380	21 & Over	0	2008	6.8
1	1	45658735	13414714	Dredd 3D	0	2012	0.0
2	2	20000000	53107035	12 Years a Slave	0	2013	8.1

	Unnamed: 0	budget	domgross	title	Response_Json	Year	imd
3	3	61000000	75612460	2 Guns	0	2013	6.7
4	4	40000000	95020213	42	0	2013	7.5

Subset the `df` DataFrame to only keep the `'domgross'`, `'budget'`, `'imdbRating'`, `'Metascore'`, and `'imdbVotes'` columns.

```
# Subset the DataFrame
df = df[['domgross', 'budget', 'imdbRating', 'Metascore', 'imdbVotes']]
df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	domgross	budget	imdbRating	Metascore	imdbVotes
0	25682380	13000000	6.8	48	206513
1	13414714	45658735	0.0	0	0
2	53107035	20000000	8.1	96	537525
3	75612460	61000000	6.7	55	173726
4	95020213	40000000	7.5	62	74170

Split the data

- First, assign the predictors to `x` and the outcome variable, `'domgross'` to `y`
- Split the data into training and test sets. Set the seed to 42 and the `test_size` to 0.25

```
# domgross is the outcome variable
X = df[['budget', 'imdbRating', 'Metascore', 'imdbVotes']]
y = df['domgross']

X_train , X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

Use the `MinMaxScaler` to scale the training set. Remember you can fit and transform in a single method using `.fit_transform()`.

```
# Transform with MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

Transform the test data (`X_test`) using the same `scaler` :

```
# Scale the test set
X_test_scaled = scaler.transform(X_test)
```

Fit a regression model to the training data

```
# Your code
linreg = LinearRegression()
linreg.fit(X_train_scaled, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

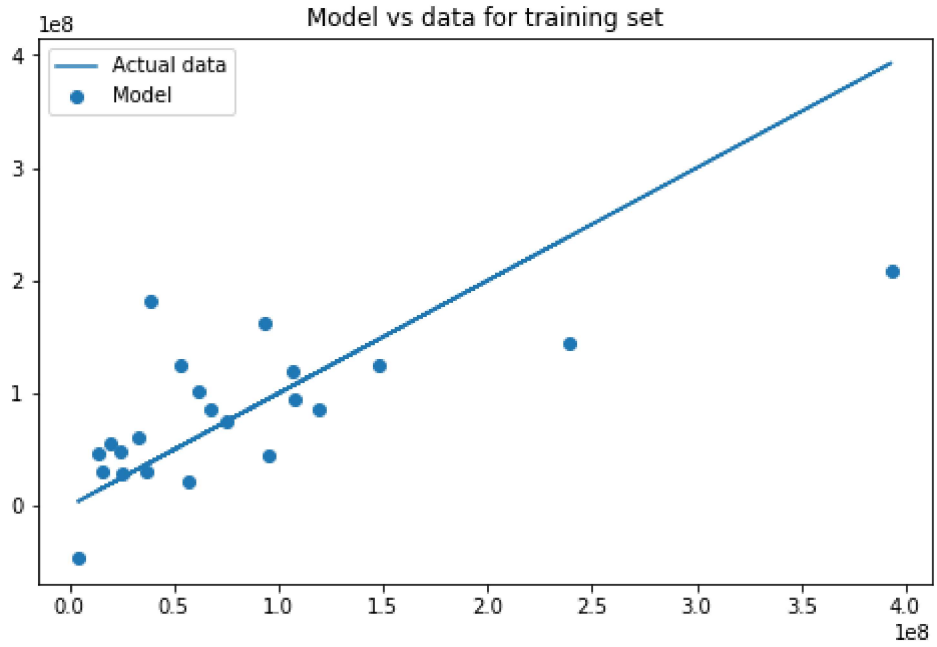
Use the model to make predictions on both the training and test sets:

```
# Training set predictions
lm_train_predictions = linreg.predict(X_train_scaled)

# Test set predictions
lm_test_predictions = linreg.predict(X_test_scaled)
```

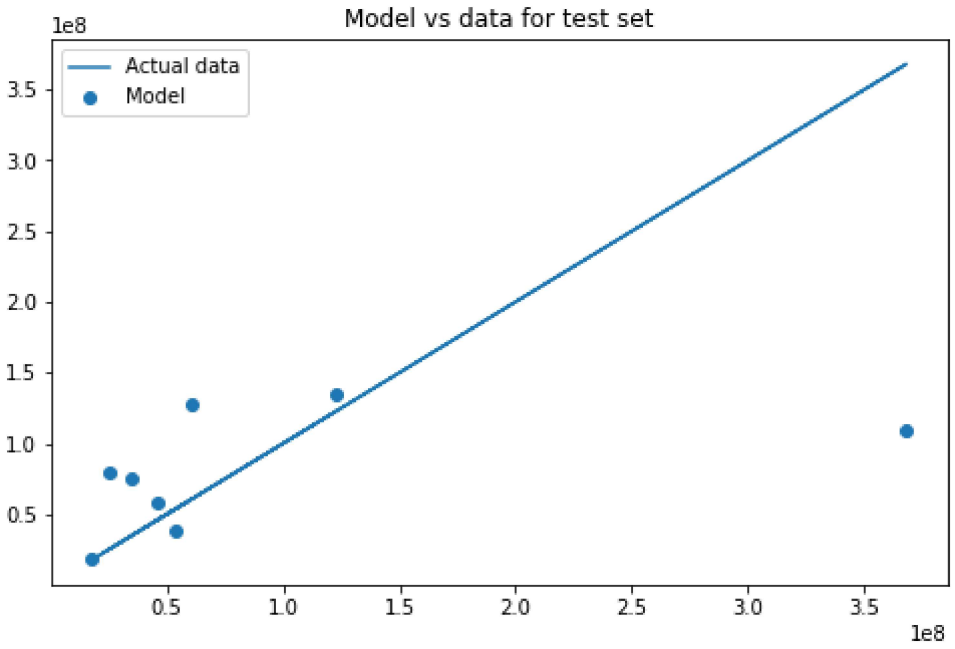
Plot predictions for the training set against the actual data:

```
# Run this cell - vertical distance between the points and the line denote the error
plt.figure(figsize=(8, 5))
plt.scatter(y_train, lm_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```



Plot predictions for the test set against the actual data:

```
# Run this cell - vertical distance between the points and the line denote the error
plt.figure(figsize=(8, 5))
plt.scatter(y_test, lm_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



Bias

Create a function `bias()` to calculate the bias of a model's predictions given the actual data: $Bias(\hat{f}(x)) = E[\hat{f}(x) - f(x)]$
(The expected value can simply be taken as the mean or average value.)

```
import numpy as np
def bias(y, y_hat):
    return np.mean(y_hat - y)
```

Variance

Create a function `variance()` to calculate the variance of a model's predictions:
$Var(\hat{f}(x)) = E[\hat{f}(x)^2] - \left(E[\hat{f}(x)]\right)^2$

```
def variance(y_hat):
    return np.mean([yi**2 for yi in y_hat]) - np.mean(y_hat)**2
```

Calculate bias and variance

```
# Bias and variance for training set
b = bias(y_train, lm_train_predictions)
```

```
v = variance(lm_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: -8.127906105735085e-09
Train variance: 3406811040986517.0
```

```
# Bias and variance for test set
b = bias(y_test, lm_test_predictions)
v = variance(lm_test_predictions)
print('Test bias: {} \nTest variance: {}'.format(b, v))
```

```
Test bias: -10982393.918069275
Test variance: 1518678846127932.0
```

Overfit a new model

Use `PolynomialFeatures` with degree 3 and transform `X_train_scaled` and `X_test_scaled`.

Important note: By including this, you don't only take polynomials of single variables, but you also combine variables, eg:

$ \text{Budget} * \text{MetaScore} ^ 2 $

What you're essentially doing is taking interactions and creating polynomials at the same time! Have a look at how many columns we get using `np.shape()`!

```
poly = PolynomialFeatures(3)

X_train_poly = poly.fit_transform(X_train_scaled)
X_test_poly = poly.fit_transform(X_test_scaled)
```

```
# Check the shape
np.shape(X_train_poly)
```

```
(22, 35)
```

Fit a regression model to the training data:

```
polyreg = LinearRegression()
polyreg.fit(X_train_poly, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

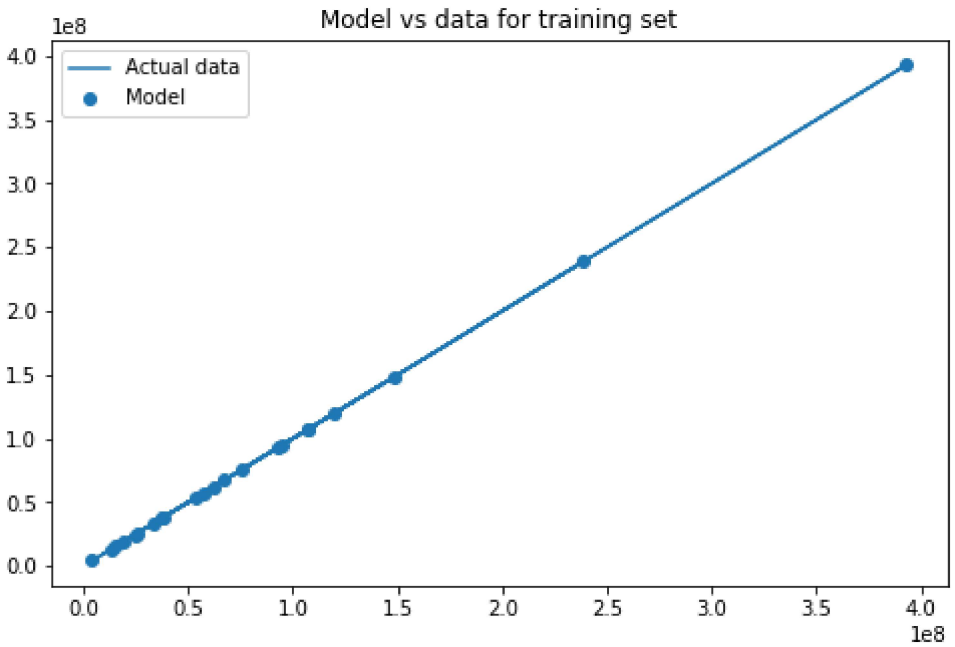
Use the model to make predictions on both the training and test sets:

```
# Training set predictions
poly_train_predictions = polyreg.predict(X_train_poly)

# Test set predictions
poly_test_predictions = polyreg.predict(X_test_poly)
```

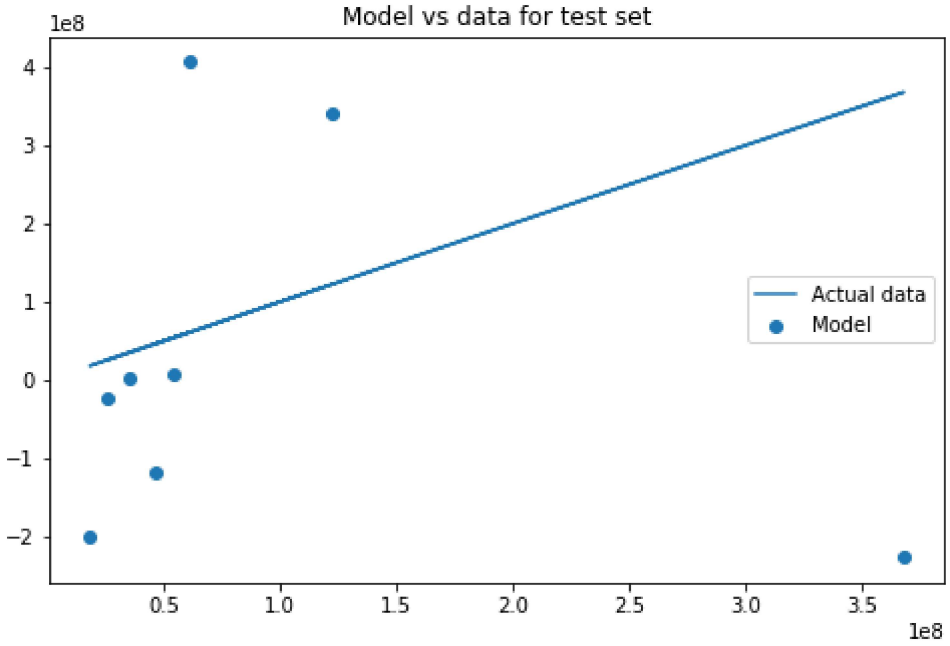
Plot predictions for the training set against the actual data:

```
# Run this cell - vertical distance between the points and the line denote the error
plt.figure(figsize=(8, 5))
plt.scatter(y_train, poly_train_predictions, label='Model')
plt.plot(y_train, y_train, label='Actual data')
plt.title('Model vs data for training set')
plt.legend();
```



Plot predictions for the test set against the actual data:

```
# Run this cell - vertical distance between the points and the line denote the error
plt.figure(figsize=(8, 5))
plt.scatter(y_test, poly_test_predictions, label='Model')
plt.plot(y_test, y_test, label='Actual data')
plt.title('Model vs data for test set')
plt.legend();
```



Calculate the bias and variance for the training set:

```
# Bias and variance for training set
b = bias(y_train, poly_train_predictions)
v = variance(poly_train_predictions)
print('Train bias: {} \nTrain variance: {}'.format(b, v))
```

```
Train bias: 3.5898251966996625e-07
Train variance: 7394168636697528.0
```

Calculate the bias and variance for the test set:

```
# Bias and variance for test set
b = bias(y_test, poly_test_predictions)
```

```
v = variance(poly_test_predictions)
print('Test bias: {} \nTest variance: {}'.format(b, v))
```

```
Test bias: -68166032.47666144
Test variance: 4.798244829435879e+16
```

Interpret the overfit model

```
# The training predictions from the second model perfectly match the actual data poi
# The bias and variance for the test set both increased drastically for this overfit
```

Level Up (Optional)

In this lab we went from 4 predictors to 35 by adding polynomials and interactions, using `PolynomialFeatures`. That being said, where 35 leads to overfitting, there are probably ways to improve by adding just a few polynomials. Feel free to experiment and see how bias and variance improve!

Summary

This lab gave you insight into how bias and variance change for a training and a test set by using both simple and complex models.

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- **Jupyter Notebook** 100.0%