Biweekly Report 1 : Linear Regression

Anthony Padilla

For the first bi-weekly report, I thought attempting to implement a simple linear regression model would assist in refreshing past knowdlege I had with ML, as well as cause me to remember key concepts that could be "untraditional" to most. I chose a wine dataset from kaggle, specifically: https://www.kaggle.com/datasets/fedesoriano/spanish-wine-quality-dataset/data I chose wine as my introduction to machine learning, and linear regression in particular ,in reference to "Predicting the Quality and Prices of Bordeaux Wine" published by O.Ashenfelter, who was an economist who predicted the price/quality of wine by using past data, and was able to do so almost precisley. He performed the feat through use of a regression model, which is why I felt it would be perfect for the first biweekly report.

Linear Regression Abstract

Linear regression is a form of analysis in where we attempt to make a linear model that can predict values based on parameters. The model of course, works "linearly", and is defined by it's intercept, and coeffecients. The processo of linear regression is to train a model to perfect, or improve, the coeffecients used in the y=mx +b equation to hopeflly make a simple, yet powerful model. Below, I will begin the process of making a linear model and performing regression, to predict things such as weighted ratings, price, as well as do an analysis of where this type of model somewhat orginated from.

In [1]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
```

In [2]:
```
#load wine dataset
data = pd.read_csv('wines_SPA.csv')

#perform data cleanup
data["year"]=data["year"].replace("N.V.",np.nan)
data=data.dropna()
data['year'] = data['year'].astype(np.int64)
data['age'] = 2024 - data["year"]   #creating new feature, 'age'

#weighing scores
data['weighted_review_score'] = data['rating']*data['num_reviews']
```

```
data['weighted_review_score'] = ( data['weighted_review_score'] / data['num_


data = data.drop(["winery","wine","country","type","num_reviews"],axis=1) #f
data.set_index("year",inplace=True)
data_2 = data #copy for second model
data.head()
```

Out[2]:

	rating	region	price	body	acidity	age	weighted_review_score
year							
2013	4.9	Toro	995.00	5.0	3.0	11	0.010639
2018	4.9	Vino de Espana	313.50	4.0	2.0	6	0.005687
2009	4.8	Ribera del Duero	324.95	5.0	3.0	15	0.322193
1999	4.8	Ribera del Duero	692.96	5.0	3.0	25	0.306379
1996	4.8	Ribera del Duero	778.06	5.0	3.0	28	0.235220

Above is a preview of the data on hand. I performed some data cleanup that would/ was causing issues, such as changing the type of year to be used as a new feature, removing an NULL or misinput entries, and dropping features that I did not believe would be useful in predicting rating. I also account for the weight of a review, as a wine with 2000 reviews and a 4.9 rating would of course be better than a wine with 1 review and a 5.0 rating.

Predicting Rating

Now is the implemnation of the model to predict rating based on : acidity, age, body, price, and region. Since region is a catergorical variable, it was one-hot encoded in order to be used for regression. It's important to note, that what is being predicted is the *weigthed* review score, as that was my goal with this model. I chose my features as I *assumed* body,age,acidity, and region would have an affect on the quality of the wine, and that price would change someones perspective on the wine (i.e it's good but it was $1000).

In [3]:

```
#one hot encode regions
data = pd.get_dummies(data,columns=['region'],drop_first=True)
X_1 = data[['acidity', 'age', 'body', 'price'] + [col for col in data.column
y_1 = data['weighted_review_score']

#split data for training and testing
X_1_train, X_1_test, y_1_train, y_1_test = train_test_split(X_1, y_1, test_s

# Perform linear regression
model_1 = LinearRegression()
model_1.fit(X_1_train, y_1_train)
```

```
# Predict values
y_1_pred = model_1.predict(X_1_test)

#Uncomment for coeffecients and intercept
#print("Coefficients:", model_1.coef_)
#print("Intercept:", model_1.intercept_)

# Uncomment for other metrics
mae = mean_absolute_error(y_1_test, y_1_pred)
mse = mean_squared_error(y_1_test, y_1_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_1_test, y_1_pred)

# Print the evaluation metrics
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

```
Mean Absolute Error (MAE): 0.031219686478902532
Mean Squared Error (MSE): 0.007677243141639173
Root Mean Squared Error (RMSE): 0.08761987868993641
R-squared (R²): 0.027085825218576165
```

Analysis of W.Rating Model

Based on our R^2 value, our features contribute very little to the weighted rating of a wine. With only a value of 0.027, this means our paramters only contribute 0.27% to changes in prediction. This could have many causes, such as insuffecent data (unlikely), wrong features, etc. Overall, this model is poor at predicing weightedrating based on our chosen parameters. It could also be that I simply am asking the wrong questions, and my approach wasn't the best. Also, while our other metrics look small, this considering the range of our expected values as well. Overall, the model did not fit the data in a predictiable way. That is to say, our model was not able to be trained in a way to give us coeffecients and an intercept that would fit this data in a frutiful way.

Predicting Price

Now we will attempt to predict price with our quality features, Ashenfelter did orginally. Hopefully, this will boast a better R^2 value, alluding to the model's feature selection to be useful.

```
In [4]: data_2.head()
```

Out[4]:

	rating	region	price	body	acidity	age	weighted_review_score
year							
2013	4.9	Toro	995.00	5.0	3.0	11	0.010639
2018	4.9	Vino de Espana	313.50	4.0	2.0	6	0.005687
2009	4.8	Ribera del Duero	324.95	5.0	3.0	15	0.322193
1999	4.8	Ribera del Duero	692.96	5.0	3.0	25	0.306379
1996	4.8	Ribera del Duero	778.06	5.0	3.0	28	0.235220

In [5]:

```
#one-hot encode once more
data_2 = pd.get_dummies(data_2,columns=['region'],drop_first=True)

#4 features to predict price
X_2 = data_2[['acidity', 'age', 'body'] + [col for col in data_2.columns if
y_2 = data_2['price']

X_2_train, X_2_test, y_2_train, y_2_test = train_test_split(X_2, y_2, test_s

# Perform linear regression
model_2 = LinearRegression()
model_2.fit(X_2_train, y_2_train)

# Predict values
y_2_pred = model_2.predict(X_2_test)

# Uncomment for coefficients and intercept
#print("Coefficients:", model_2.coef_)
#print("Intercept:", model_2.intercept_)

# Uncomment for other eval.metrics
mae_2 = mean_absolute_error(y_2_test, y_2_pred)
mse_2 = mean_squared_error(y_2_test, y_2_pred)
rmse_2 = np.sqrt(mse_2)
r2_2 = r2_score(y_2_test, y_2_pred)

print(f"Mean Absolute Error (MAE): {mae_2}")
print(f"Mean Squared Error (MSE): {mse_2}")
print(f"Root Mean Squared Error (RMSE): {rmse_2}")
print(f"R-squared (R²): {r2_2}")
print(y_2_test)
print(y_2_pred)
```

```
Mean Absolute Error (MAE): 58.058839110797265
Mean Squared Error (MSE): 24475.22838680931
Root Mean Squared Error (RMSE): 156.44560839732546
R-squared (R²): 0.20854166230592508
year
2016     19.98
2017     24.45
2015     17.50
2017     18.90
2014     77.36
         ...
2016     48.96
2011     61.94
2011     28.53
2011     51.35
2017    106.00
Name: price, Length: 1214, dtype: float64
[-13.04795994  24.45        -0.80223445 ...  48.18066749 128.34983745
 -25.29368542]
```

Analysis Of Second Model

Comparing the model to the previous one, it does seem as our paramters are more effective at predicting price rather than rating. The R^2 value of ~21% suggest that our parameters are responsible for 21 % of the change in values, which is more than what it was contributuing when predicting rating. However, it isn't nearly as good as we would like it to be, and wouldn't be valuable as a way to predict prices. So, based on our features and data, we really can't predict much/anything useful. The other performance metrics don't raise hope either, as the error's aren't neglible.

Revisiting Ashenfelter

Since the data/features used in the data set were fruitless, I decided to close off the report by revisiting Ashenfelter's orginal model for predicting wine prices. Ashenfelter was able to accuratley predict wine prices by using the following equation : *log(price) = intercept + c_1(avg. summer temp) + c_2(winter_rainfall) + c_3 (harvest_rainfall) + c_4 (age of wine)*. With this equation, he was able to accuratley predict the quality of the wine (where we are using log(price)). I will replicate the model used, which is very popular and can be found online, but will also seperated some of the data to see what the R^2 value of these feaures were.

In [6]:
```
#import
wine_df = pd.read_csv('bordeaux.csv',index_col='year')
#seperate data
wine_test = wine_df.loc[1981:].copy()
wine_train = wine_df.loc[:1980].copy()
```

```
#log(price) = quality
wine_train['log(price)'] = np.log(wine_train['price'])

#train on features
wine_model = LinearRegression()
wine_model.fit(X=wine_train[['summer','win','har','age']], y=wine_train['log

wine_predict = wine_model.predict(X=wine_test[['summer','win','har','age']])

print("Wine Model Coeffecients:",wine_model.coef_)
print("Intercept:",wine_model.intercept_)
```

```
Wine Model Coeffecients: [ 0.61871092  0.00119721 -0.00374825  0.02435187]
Intercept: -7.831137841446709
```

Ashenfelters original model had the coeffecients *quality = 0.62(avg. summer temp) + 0.0012(winter rainfall) -0.0037 (harvest rainfall) + 0.024(age of wine) - 7.83* , which are the coeffecients we've replicated here. The effectivness of the is model to predict future prices/quality of wine can come from many things, but the important one to note in this exercise is the features. Compared to the data set I used, these features were more telling and crucial to the quality of the wine.

R^2 Of Ashenfelters Model

Using a 7 to 1 training test ratio, I will alter only the data to see performance metrics for the orginal model.

In [7]:
```
wine_test = wine_df.loc[1976:1980].copy()
wine_train = wine_df.loc[:1975].copy()



#log(price) = quality
wine_train['log(price)'] = np.log(wine_train['price'])

#train on features
wine_model = LinearRegression()
wine_model.fit(X=wine_train[['summer','win','har','age']], y=wine_train['log

wine_predict = wine_model.predict(X=wine_test[['summer','win','har','age']])

print("Wine Model Coeffecients:",wine_model.coef_)
print("Intercept:",wine_model.intercept_)

mae_wine = mean_absolute_error(np.log(wine_test['price']),wine_predict)
mse_wine = mean_squared_error(np.log(wine_test['price']), wine_predict)
rmse_wine = np.sqrt(mse_wine)
#print(wine_predict)
#print(np.log(wine_test['price']))
r2_wine = r2_score(np.log(wine_test['price']),wine_predict)
```

```
print("R^2 value:",{r2_wine})
print(wine_model.predict([[18.7,468,80,31]]))
print(f"Mean Absolute Error (MAE): {mae_wine}")
print(f"Mean Squared Error (MSE): {mse_wine}")
print(f"Root Mean Squared Error (RMSE): {rmse_wine}")
```

```
Wine Model Coeffecients: [ 0.55092207  0.00134516 -0.00413456  0.02539487]
Intercept: -6.771591224636763
R^2 value: {0.15847180954743045}
[4.61666066]
Mean Absolute Error (MAE): 0.2741152786703389
Mean Squared Error (MSE): 0.10058031938989667
Root Mean Squared Error (RMSE): 0.31714400418405625
```

```
/home/hakarikinji/jupyter/lib/python3.11/site-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but LinearRegression was f
itted with feature names
  warnings.warn(
```

Summary

The creation of a linear reggression model that can accuratley predict some desired feature is not as simple as calling python libraries. Multiple things need to be kept in consideation, such as the amount of data available, the quality of data, the impact of features, and it doesn't end there. Linear regression is powerful tool, but there are important things to take into account, such as if our data even works in a linear fasion. Based off of the data set I used, we weren't able to make a effecient model. Looking at submitted code on kaggle regarding the data set, it appears others have ran far more conclusive tests, but came to a similiar conclusion. There exist other datasets that include different features, such as actual chemical makeup. The model I made used features that would be byproducts of other causes. Comparing this to Ashenfelters original model, that took into account rain+temp+age, it was clear these features were more telling of the "price" of a wine than the ones I used.

I would also like to note that in my test of these models, I refreshed myself on the imporantce of understanding how to evaluate a model. When comparing R^2 values between Ashenfelters mdoel and the one I made, there are somewhat similar. However, their accuracies are of course completley different, and when comparing other perfomrance metrics, it provides more insight that of course my model wasn't very good.

Overall, the creation of a mathematical model is one that requires careful attention, understanding of the problem, iteration, repetition, and data. This biweekly report served as refresher on past knowledge, as well as more experience working with simple models.