hw3

October 13, 2024

1 HW 3 - Ishaan Sathaye

1.1 Section A: Math

1. Express this model in matrix form. For each matrix involved in the equation, state the dimensions, and what those dimensions represent.

- $Y = \begin{bmatrix} 1,000,000 \\ 400,000 \\ 700,000 \end{bmatrix}$

- where the dimensions are 3x1 and 3 rows represents the number of observations and the 1 column represents the house price.

- $X = \begin{bmatrix} 1 & 4700 & 3 \\ 1 & 1050 & 1 \\ 1 & 2200 & 2 \end{bmatrix}$

- where the dimensions are 3x3 and 3 rows represents the number of observations and the 3 columns represent the intercept, square footage, and number of bedrooms.

- $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$

- where the dimensions are 3x1 and 3 rows represents the number of coefficients (including intercept) and the 1 column represents the coefficient values.

- Matrix Form:

  - $Y = X\beta$
  - $\begin{bmatrix} 1,000,000 \\ 400,000 \\ 700,000 \end{bmatrix} = \begin{bmatrix} 1 & 4700 & 3 \\ 1 & 1050 & 1 \\ 1 & 2200 & 2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$

2. Compute the hat matrix for this data. Explain why you got the results you did.

```
[2]: import numpy as np

     def hat_matrix(X):
         H = X @ np.linalg.inv(X.T @ X) @ X.T
         return H
```

```python
X = np.array([[1, 4700, 3], [1, 1050, 1], [1, 2200, 2]])
H = hat_matrix(X)
print(H)
```

```
[[1.00000000e+00 0.00000000e+00 4.44089210e-16]
 [7.10542736e-15 1.00000000e+00 5.32907052e-15]
 [1.06581410e-14 5.32907052e-15 1.00000000e+00]]
```

- For the hat matrix, I got a identity matrix with dimensions 3x3, 0s everywhere but the diagonal. The hat matrix should be a identity matrix because the model is a perfect fit. The model is a perfect fit because the number of observations is equal to the number of coefficients.

3. Express this model in matrix form. For each matrix involved in the equation, state the dimensions, and what those dimensions represent.

- $Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}$

- where the dimensions are nx1 and n rows represents then number of strains in the dataset and the 1 column represents the rating of each strain.

- $X = \begin{bmatrix} 1 & X_{11} & X_{12} & \dots & X_{163} \\ 1 & X_{21} & X_{22} & \dots & X_{263} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & Xn63 \end{bmatrix}$

- where the dimensions are nx164 and n rows represents the number of strains in the dataset and the 64 columns represent the intercept and the 63 columns for the dummy variables representing the effects and flavors of the strains.

- $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{63} \end{bmatrix}$

- where the dimensions are 64x1 and 64 rows represent the intercept nad coefficients for each of the 63 dummy variables and the 1 column represents the coefficient values.

- $\epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$

- where the dimensions are nx1 and n rows represents the residuals for each strain and the 1 column represents the residual values.

- Matrix Form:
  - $Y = X\beta + \epsilon$
  - $\begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} & \dots & X_{163} \\ 1 & X_{21} & X_{22} & \dots & X_{263} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{n63} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{63} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$

4. Give the *equation* for the hat matrix for this model using a OLS. Where would you expect this equation to "break"?

- $H = X(X^TX)^{-1}X^T$
- The equation would break if the matrix X^TX is not invertible. This would happen if any of the columns are highly correlated or linearly dependent. X^TX would be singular and so the inverse would not exist.
- The equation would also break if the number of observations is less than the number of coefficients. This would make the matrix X^TX singular and the inverse would not exist. However from the csv file, I know that this is not the case.

5. Give the *equation* for the hat matrix for this model using a *Ridge penalty*.

- $H = X(X^TX + \lambda I)^{-1}X^T$

6. Find the equation for the OLS hat matrix for this model predicting residuals. How does it compare to your hat matrix in A2?

- $H = X(X^TX)^{-1}X^T$
- The hat matrix for predicting residuals is identical to the hat matrix in A2 and this is because the same design matrix and OLS formula is used. But this hat matrix is used to essentially map the observed residuals R to the predicted residuals $\hat{R}$.

7. Would you recommend this strategy? Why or why not? Justify your answer mathematically.

- I would not recommend this strategy because if you are predicting further from the design matrix, it would result in redundant predictions.
- $R = Y - H_1Y$, where H_1 is the hat matrix for predicting Y and orthogonal to the design matrix.
- So fitting a model to predict the residuals using the same design matrix is ineffective:
- $H_2R = X(X^TX)^{-1}X^TR = 0$
- This is because the residuals are orthogonal to the design matrix and so the predicted residuals would be 0.
- So if $\hat{R} = 0$, then the predicted house prices will be the same as the first model's predictions:
- $\hat{Y} = H_1Y - H_2R = H_1Y - 0 = H_1Y$

1.2 Section B: Programming

1. Read the data in, and drop all rows with missing data in the predictors. Decide if you want to standardize anything or not.

```
[38]: import pandas as pd

df = pd.read_csv("./cannabis_full.csv")
predictors = df.drop(columns=['Strain', 'Type', 'Effects', 'Flavor', 'Rating'])
df_clean = df.dropna(subset=predictors.columns)
df_clean.head()
```

```
[38]:            Strain     Type  Rating                                     Effects  \
      0           100-Og  hybrid     4.0  Creative,Energetic,Tingly,Euphoric,Relaxed
      1  98-White-Widow  hybrid     4.7      Relaxed,Aroused,Creative,Happy,Energetic
```

```
2                1024   sativa    4.4    Uplifted,Happy,Relaxed,Energetic,Creative
3            13-Dawgs   hybrid    4.2      Tingly,Creative,Hungry,Relaxed,Uplifted
4            24K-Gold   hybrid    4.6    Happy,Relaxed,Euphoric,Uplifted,Talkative

                          Flavor  Creative  Energetic  Tingly  Euphoric  Relaxed  \
0           Earthy,Sweet,Citrus       1.0        1.0     1.0       1.0      1.0
1         Flowery,Violet,Diesel       1.0        1.0     0.0       0.0      1.0
2      Spicy/Herbal,Sage,Woody       1.0        1.0     0.0       0.0      1.0
3    Apricot,Citrus,Grapefruit       1.0        0.0     1.0       0.0      1.0
4           Citrus,Earthy,Orange      0.0        0.0     0.0       1.0      1.0

      …  Ammonia  Minty  Tree  Fruit  Butter  Pineapple  Tar  Rose  Plum  Pear
0    …      0.0    0.0   0.0    0.0     0.0        0.0   0.0   0.0   0.0   0.0
1    …      0.0    0.0   0.0    0.0     0.0        0.0   0.0   0.0   0.0   0.0
2    …      0.0    0.0   0.0    0.0     0.0        0.0   0.0   0.0   0.0   0.0
3    …      0.0    0.0   0.0    0.0     0.0        0.0   0.0   0.0   0.0   0.0
4    …      0.0    0.0   0.0    0.0     0.0        0.0   0.0   0.0   0.0   0.0

[5 rows x 69 columns]
```

2. Write a function to compute beta estimates for Ordinary Least Squares regression with multiple predictors. Run this function on the Cannabis data with three predictors: Type, Relaxed, and Energetic.

```python
[39]:  # Write a function to compute beta estimates for Ordinary Least Squares
       # regression with multiple predictors. Run this function on the Cannabis data
       # with three predictors: Type, Relaxed, and Energetic.
       def compute_betas(X, y):
           betas = np.linalg.inv(X.T @ X) @ X.T @ y
           return betas


       df_clean = pd.get_dummies(df_clean, columns=['Type'], drop_first=True)
       X = df_clean[['Type_indica', 'Type_sativa', 'Relaxed', 'Energetic']].values
       X = np.hstack((np.ones((X.shape[0], 1)), X))
       y = df_clean['Rating'].values
       betas = compute_betas(X, y)
       print(betas)
```

```
[3.74369181 0.0246247  0.06176504 0.61677751 0.3575477 ]
```

3. Write a function to compute beta estimates for Ridge regression, with λ as a user input. Use the following loss function:

- $l(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

```python
[40]:  def compute_betas_ridge(X, y, lam):
           betas = np.linalg.inv(X.T @ X + (X.shape[0] * lam) * np.eye(X.shape[1])) @
       X.T @ y
```

```
        return betas
```

4. Run your Ridge function on the Cannabis data with three predictors: Type, Relaxed, and Energetic with $\lambda = 1000$. How do these estimates compare to your OLS estimates?

```
[41]: betas_ridge = compute_betas_ridge(X, y, 1000)
      print(betas_ridge)
```

```
[0.0043112  0.00129269 0.00080465 0.00329303 0.00120866]
```

- The OLS estimates are higher than the Ridge estimates. This is because the Ridge estimates are penalized by the λ term and so the coefficients are shrunk towards 0.

5. Run your Ridge function on the Cannabis data with all the predictors with $\lambda = 1000$.

```
[44]: # 5. Run your Ridge function on the Cannabis data with all the predictors with
      ↪$\lambda = 1000$.

      predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor', 'Rating'])
      X = predictors.values
      X = np.hstack((np.ones((X.shape[0], 1)), X))
      betas_ridge = compute_betas_ridge(X, y, 1000)
      print(betas_ridge)
```

```
[4.29985189e-03 1.40274876e-03 1.20561358e-03 6.40154628e-04
 3.08487327e-03 3.28431008e-03 3.77670869e-04 3.54424974e-03
 2.84077793e-03 8.78311533e-04 6.78176397e-04 5.49476789e-04
 1.12441374e-03 1.38053514e-03 1.73257123e-06 1.73257123e-06
 2.10751664e-03 2.02122803e-03 1.00950921e-03 5.13166810e-04
 1.37021178e-05 4.59673992e-04 4.33758626e-04 7.57394052e-05
 4.79311197e-04 1.71994242e-05 7.25778317e-05 1.48435658e-04
 8.61025001e-04 3.13815595e-04 6.48992150e-04 3.33098052e-04
 6.85746076e-04 1.11409843e-04 4.51563533e-05 2.91136421e-04
 1.19091611e-04 7.08955773e-05 6.13422437e-05 3.64210109e-04
 1.11526604e-05 6.49251687e-05 4.78697692e-05 1.24918107e-05
 3.18412027e-05 1.64985999e-05 2.95485558e-04 8.93771969e-05
 2.78652659e-04 1.06391017e-04 2.85558389e-05 6.01444004e-05
 7.14140539e-05 1.03959264e-04 4.62779443e-05 5.43182314e-05
 8.15461080e-05 6.44130307e-05 6.44130307e-05 3.57667882e-05
 8.11192791e-05 1.51631925e-05 3.04571745e-05 3.79965845e-06
 5.74766318e-06 1.28929182e-03 8.02552982e-04]
```

6. Write a function to perform *tuning* on λ, using cross-validation.

```
[52]: def tune_lambda_split(train, test, lam_values, metric):
          X_train = train.drop(columns=['Rating']).values
          X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
          y_train = train['Rating'].values
          X_test = test.drop(columns=['Rating']).values
```

```python
        X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
        y_test = test['Rating'].values
        metrics = []
        for lam in lam_values:
            betas = compute_betas_ridge(X_train, y_train, lam)
            y_pred = X_test @ betas
            if metric == 'r-sq':
                y_bar = np.mean(y_test)
                ss_tot = np.sum((y_test - y_bar) ** 2)
                ss_res = np.sum((y_test - y_pred) ** 2)
                r2 = 1 - ss_res / ss_tot
                metrics.append(r2)
            elif metric == 'mse':
                mse = np.mean((y_test - y_pred) ** 2)
                metrics.append(mse)
            elif metric == 'mae':
                mae = np.mean(np.abs(y_test - y_pred))
                metrics.append(mae)
        return pd.DataFrame({'lambda': lam_values, metric: metrics})

def tune_lambda(df, lam_values, metric, k):
    n = df.shape[0]
    fold_size = n // k
    metrics = []
    for lam in lam_values:
        metric_values = []
        for i in range(k):
            test = df.iloc[i * fold_size:(i + 1) * fold_size]
            train = df.drop(test.index)
            df_metric = tune_lambda_split(train, test, [lam], metric)
            metric_values.append(df_metric[metric].values[0])
        metrics.append(np.mean(metric_values))
    return pd.DataFrame({'lambda': lam_values, metric: metrics})
```

7. Use your tuning function to choose a best λ for Ridge regression.

```python
[54]: predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor'])
      lam_values = np.logspace(0, 5, num=100)
      df_tune = tune_lambda(predictors, lam_values, 'r-sq', 5)
      best_lambda_rsq = df_tune.loc[df_tune['r-sq'].idxmax()]['lambda']
      print(best_lambda_rsq)

      df_tune = tune_lambda(predictors, lam_values, 'mse', 5)
      best_lambda_mse = df_tune.loc[df_tune['mse'].idxmin()]['lambda']
      print(best_lambda_mse)

      df_tune = tune_lambda(predictors, lam_values, 'mae', 5)
```

```python
best_lambda_mae = df_tune.loc[df_tune['mae'].idxmin()]['lambda']
print(best_lambda_mae)
```

1.0
1.0
1.0

- Seems that $\lambda = 1$ is the best λ for Ridge regression.