hw3

October 15, 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.

[258]:
```
import numpy as np

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

```
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.
- $X = \begin{bmatrix} 1 & X_{11} & X_{12} & ... & X_{163} \\ 1 & X_{21} & X_{22} & ... & X_{263} \\ \vdots & \vdots & \vdots & ... & \vdots \\ 1 & X_{n1} & X_{n2} & ... & Xn63 \end{bmatrix}$
- 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:
  - $Y = X\beta + \epsilon$
  - $\begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} & ... & X_{163} \\ 1 & X_{21} & X_{22} & ... & X_{263} \\ \vdots & \vdots & \vdots & ... & \vdots \\ 1 & X_{n1} & X_{n2} & ... & X_{n63} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{63} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$

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 it is 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.

[259]:
```
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()
```

[259]:

	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.

[260]:
```
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:

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

[261]:
```
def compute_betas_ridge(X, y, lam):
    n, p = X.shape
    betas = np.linalg.inv(X.T @ X + lam * np.eye(p)) @ 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?

[262]:
```
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_ridge = compute_betas_ridge(X, y, 1000)
print(betas_ridge)
```

```
[2.09146146 0.42927685 0.37862156 1.29022622 0.59328568]
```

- 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 $\lambda = 1000$.

[263]:
```
predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor', 'Rating'])
X = predictors.values
betas_ridge = compute_betas_ridge(X, y, 1000)
print(betas_ridge)
```

```
[4.80029634e-01 4.20427905e-01 2.70827248e-01 7.17088330e-01
 8.11775732e-01 2.06998827e-01 8.43333169e-01 7.00186100e-01
 2.95768601e-01 3.05102280e-01 2.22007493e-01 3.94345823e-01
 3.74779752e-01 3.56266939e-03 3.56266939e-03 4.21006427e-01
 4.33017084e-01 2.75309045e-01 1.66117427e-01 8.59336419e-03
 1.57083178e-01 1.67099289e-01 3.22719662e-02 1.11911001e-01
 8.59998596e-03 2.56604438e-02 5.21142541e-02 2.10353024e-01
 1.02429552e-01 1.79572463e-01 1.15369591e-01 1.89508957e-01
 5.91956879e-02 2.29877832e-02 9.22359824e-02 4.08288775e-02
 2.91941620e-02 2.18929490e-02 1.24655405e-01 8.58042452e-03
 2.34630359e-02 2.09105933e-02 9.19884385e-03 1.31976747e-02
 4.98400069e-03 1.02501134e-01 3.16569077e-02 8.76123475e-02
 5.27809201e-02 1.16266605e-02 2.76326947e-02 2.96246338e-02
 4.54533066e-02 2.30035231e-02 2.76335381e-02 4.15639678e-02
 2.41932056e-02 2.41932056e-02 1.32495822e-02 2.34543245e-02
 6.22272058e-03 1.25641352e-02 5.14925575e-04 3.23613676e-03
 2.94807678e-01 2.42561359e-01]
```

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

[264]:
```
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
    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.

[265]:
```
predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor'])
lam_values = np.logspace(0, 5, num=100)
folds = 5
df_tune_rsq = tune_lambda(predictors, lam_values, 'r-sq', folds)
best_lambda_rsq = df_tune_rsq.loc[df_tune_rsq['r-sq'].idxmax()]['lambda']
print(best_lambda_rsq)

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

df_tune_mae = tune_lambda(predictors, lam_values, 'mae', folds)
best_lambda_mae = df_tune_mae.loc[df_tune_mae['mae'].idxmin()]['lambda']
print(best_lambda_mae)
```

```
16.297508346206442
16.297508346206442
11.497569953977356
```

- Seems that $\lambda = 16.29$ is the best λ for Ridge regression as it got the highest R-sq and the lowest mse, and $\lambda = 11.49$ for the lowest mae.

1.3 Section C: Concepts

1. The documentation for the linear_model.Ridge function in scikit-learn here. The argument alpha is the penalty parameter here - but the loss function is parametrized differently than in B3. What would you have to plug in for alpha to get the same results as you got in B5?
   - The loss function in scikit-learn is parametrized as:
     - $||y - Xw||_2^2 + \alpha||w||_2^2$
   - While the loss function in B3 is parametrized as:
     - $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$
   - To get the same results as in B5, you would just plug in 1000*n for alpha, where n is the number of observations.
2. In tidymodels, the penalty argument of linear_reg() is the penalty parameter. If we use the glmnet package as our engine, penalty gets passed to the argument lambda of the function glmnet. Read the documentation for the glmnet() function - scroll down to the "Details" section. What would you have to plug in for penalty to get the same results as you got in B5?
   - Plug in 2 times the λ times the number of observations for penalty to get the same results as in B5.
3. Fit your final model from B7, and interpret the three largest coefficients.

[267]:
```
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, best_lambda_rsq)

predictor_names = predictors.columns
coefficients_with_names = list(zip(betas_ridge, ['Intercept'] +␣
 ↪list(predictor_names)))
largest_coefficients = sorted(coefficients_with_names, key=lambda x: abs(x[0]),␣
 ↪reverse=True)[:3]
print("Largest 3 coefficients:")
for coef, name in largest_coefficients:
    print(f"{name}: {coef}")
```

```
Largest 3 coefficients:
Intercept: 1.640762409963628
Relaxed: 0.6013594958352564
Creative: 0.48773022896942014
```

- Interpretation:
  - For every 1 unit increase in the Happy rating, the rating of the strain increases by 0.617.
  - For every 1 unit increase in the Relaxed rating, the rating of the strain increases by 0.579.
  - The intercept is the rating of the strain when all the predictors are 0, which is not

interpretable in this context.

4. Use built-in functions from scikit-learn or tidymodels to fit a LASSO Regression, using all the predictors. Use the same value of that you found to be "best" in B5. Interpret the largets three coefficients - are they the same as in C1?

[285]:
```
from sklearn.linear_model import Lasso

predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor', 'Rating'])
X = predictors.values
y = df_clean['Rating'].values

lasso = Lasso(alpha=best_lambda_rsq)
lasso.fit(X, y)
coefficients_with_names = list(zip(lasso.coef_, predictors.columns))
largest_coefficients = sorted(coefficients_with_names, key=lambda x: abs(x[0]),␣
 ↪reverse=True)[:3]
print("Largest 3 coefficients:")
for coef, name in largest_coefficients:
    print(f"{name}: {coef}")
```

```
Largest 3 coefficients:
Creative: 0.0
Energetic: 0.0
Tingly: 0.0
```

- Interpretation:
  - The coefficients are 0 because LASSO regression shrinks the coefficients to 0 and it would seem that the lambda value is too high. The coefficients are not same as in the previous question.

5. In which of these model fitting approaches (OLS, Ridge, LASSO) was it important to standardize our predictors before fitting the model? Why?

- It is important to standardize the predictors before fitting the model in Ridge and LASSO regression because the penalty term is dependent on the scale of the predictors. If the predictors are not standardized, then the penalty term would be biased towards the predictors with larger scales.

6. Which of the three model fitting approaches do you prefer in this data scenario? Why? (There is no single right answer to this question; I want you to justify your choice in a reasonable way.)

- I prefer Ridge regression in this data scenario because it is a good compromise between OLS and LASSO regression. Ridge regression is able to shrink the coefficients towards 0 and so it is able to handle multicollinearity and overfitting. LASSO regression is too harsh in shrinking the coefficients to 0 and so it is not able to handle multicollinearity as well as Ridge regression.