NYU CS-GY 6923: Machine Learning Fall 2024

Homework 1

Name: Rohan Mahesh Gore (N19332535)

Problem 1. Minimizing a Weighted Loss Function.

Collaborators: None.

As we have the standard multivariate linear model under consideration, of the form

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \langle \boldsymbol{\beta}, \mathbf{x} \rangle$$

We are interested in minimizing the WSS (Weighted Sum of Squares) Loss as it is very useful when data is procured from different locations/instruments where, some of those come are from a more accurate and a reliable location/instrument.

We are interested in minimizing the weighted sum-of-squares loss:

$$L_{WSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} w_i \cdot (\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle - y_i)^2$$

a. Deriving the expression for the gradient of $L_{WSS}(\boldsymbol{\beta})$

Given the weighted sum-of-squares loss function, we can derive the expression for the gradient in few steps:

1. Expand the Loss Function:
First, we expand the squared term to make it more easier for performing differentiation.

$$L_{WSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} w_i \cdot \left(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle^2 - 2y_i \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle + y_i^2\right)$$

2. Differentiation with respect to each component of $\boldsymbol{\beta}$, denoted as β_j.

Now, we take the partial derivative with respect to β_j.

$$\frac{\partial L_{WSS}}{\partial \beta_j} = \sum_{i=1}^{n} w_i \cdot \frac{\partial}{\partial \beta_j} \left(\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle^2 - 2y_i \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle + y_i^2\right)$$

Since y_i^2 is not depend on β_j, its derivative is zero. Therefore, we have two terms left, which can be differentiated by using the chain rule of derivations.

For the first term, $\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle^2$, applying the chain rule gives:

$$\frac{\partial}{\partial \beta_j} \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle^2 = 2\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle \cdot \frac{\partial}{\partial \beta_j} \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle = 2\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle \cdot x_{i,j}$$

For the second term, $-2y_i\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle$, the derivative will be:

$$\frac{\partial}{\partial \beta_j} - 2y_i \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle = -2y_i x_{i,j}$$

3. Substituting the derivation back into the original equation:

$$\frac{\partial L_{WSS}}{\partial \beta_j} = \sum_{i=1}^{n} w_i \cdot (2\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle x_{i,j} - 2y_i x_{i,j})$$

4. Final Simplified form:

$$\frac{\partial L_{WSS}}{\partial \beta_j} = 2 \sum_{i=1}^{n} w_i \cdot (\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle - y_i)\, x_{i,j}$$

Therefore, the expression of gradient for $\nabla L_{WSS}(\boldsymbol{\beta})$ is:

$$\nabla L_{WSS}(\boldsymbol{\beta}) = 2 \sum_{i=1}^{n} w_i \cdot (\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle - y_i) \cdot \mathbf{x}_i$$

Hence, Derived.

b. Deriving the closed form expression for $\boldsymbol{\beta}^*$

To find $\boldsymbol{\beta}^*$, we set the gradient expression, derived in the first part of the problem, to zero and solve further for $\boldsymbol{\beta}$.

1. Set the Gradient to Zero:

$$2 \sum_{i=1}^{n} w_i \cdot (\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle - y_i)\, x_{i,j} = 0$$

2. This equation can be rewritten in matrix form as follows:

$$2\mathbf{X}^T \mathbf{W}(\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) = 0$$

where,
- $\mathbf{W}$ is a diagonal matrix with w_i on the diagonal,
- $\mathbf{X}$ is the matrix of input vectors, and
- $\mathbf{y}$ is the vector of target values.

3. By solving for $\boldsymbol{\beta}$ we get:

$$\boldsymbol{\beta}^* = (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}$$

Therefore, this is the closed-form expression for $\boldsymbol{\beta}^*$ that minimizes the weighted sum-of-squares loss function $L_{WSS}(\boldsymbol{\beta})$.

Problem 2. Machine Learning Does Averages.

Collaborators: None.

a. Arithmetic mean

Given the loss function $L(m) = \sum_{i=1}^{n}(y_i - m)^2$, we need to show that $L(m)$ is minimized when m is the arithmetic mean. This can be shown in few steps as follows:

1. Expand the Loss Function:
First, we expand the squared term to make it more easier for performing differentiation.

$$L(m) = \sum_{i=1}^{n}\left(y_i^2 - 2my_i + m^2\right)$$

2. Differentiation $L(m)$ with respect to m:

$$\frac{\partial L}{\partial m} = \sum_{i=1}^{n}(-2y_i + 2m)$$

3. Setting the Derivative to Zero:
Setting the computed derivative to zero and solving it for m gives:

$$\sum_{i=1}^{n}(-2y_i + 2m) = 0$$

$$2m \cdot n = 2\sum_{i=1}^{n} y_i$$

$$m = \frac{1}{n}\sum_{i=1}^{n} y_i = \bar{y}$$

Therefore, we have successfully demonstrated that the that the loss function $L(m)$ can be minimized by setting m as the arithmetic mean.

b. Minimizing Maximum Absolute Deviation

The loss fucntion that we are trying to minimize is the maximum absolute deviation, which measures the maximum absolute deviation between m and any data point y_i. It is given by:

$$L(m) = \max_i |y_i - m|$$

To minimize this loss, we need to find the value of $'m'$ such that it minimizes the given loss function. This can be achieved when m is itself a value that is most representation of the dataset.

That is, m needs to be a measure of central tendency: mean, meadian, mode or midrange.

We can consider the working of each of the above mentioned measures of central tendency, in order to find out which will minimize the given loss function.

1. Mean:

- The mean is the average of all the data points, and it can minimize the sum of squared deviations but not the maximum absolute deviation.
- Also, mean can be pulled towards extreme values due to outliers in the data.
- As a result, the mean might not be in the middle of the data, and the maximum distance between the mean and the furthest data points can be quite large.
- Hence, mean doesn't minimize the maximum absolute deviation.

2. Median:

- The median of a dataset is the middle value when the data is sorted. Specifically:
  - If the number of data points is odd, the median is the middle value.
  - If the number of data points is even, the median is the average of the two middle values.
- The median is a great option for minimizing our loss function because it splits the dataset into two equal halves. This makes the largest distance between the median and the data points pretty small.
- Though it reduces the deviation, it does not take into account the extreme values in the distribution.
- Thus, median is not the one which will minimize our loss function as it is successful in minimizing absolute deivation but not the maximum absolute deviation.

3. Mode: The mode is the value that appears most frequently in the dataset. However, the mode says nothing about the distribution of the data around it and hence, is not the correct representative of the data.

4. Midrange:

- The midrange is the average of the minimum and maximum values in the dataset and is given by:
$$\text{Midrange} = \frac{\min(y) + \max(y)}{2}$$
- Midrange is optimal due to its equidistant property from the minimum and maximum values in the dataset. By choosing a point exactly in the middle of the extreme values, we ensure that no point in the dataset can be further away than half the range of the data.

Mathematical proof for Midrange:

Let m* be the midrange. For any data point y_i:

$$|y_i - m*| \leq (y_m ax - y_m in)/2$$

This is true because y_i is between $min(y)$ and $max(y)$, and $m*$ is in the middle.
For any other choice of m:

$$max_i |y_i - m| \geq max(|max(y) - m|, |min(y) - m|) \geq (max(y) - min(y))/2$$

Therefore, the midrange minimizes the maximum absolute deviation.

Though, like median it does not account for the major chunk of distribution it performs well in practice. Even-though it is sensitive to outliers, it handles the loss in randomly generated distributions efficiently, and minimizes the loss function.

Practical Implementation: We have considered the value m to be mean, median, mode and midrange. Then we have plotted the overall losses. This is done for a randomly generated simple distribution and on a skewed distribution.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats


# Calculate mean, median, mode, and midrange
def calc (data):
  mean = np.mean(data)
  median = np.median(data)
  mode = stats.mode(data, keepdims=False)[0]  # Extract mode directly
  midrange = (np.min(data) + np.max(data)) / 2
  return mean, median, mode, midrange

def max_absolute_deviation(data, m):
    return np.max(np.abs(data - m))

# Calculate loss for each measure
def cal_loss(data, mean, median, mode, midrange):
  mean_loss = max_absolute_deviation(data, mean)
  median_loss = max_absolute_deviation(data, median)
  mode_loss = max_absolute_deviation(data, mode)
  midrange_loss = max_absolute_deviation(data, midrange)
  return mean_loss, median_loss, mode_loss, midrange_loss

# Visualizing the result

def plotter (mean_loss, median_loss, mode_loss, midrange_loss):
  measures = ['Mean', 'Median', 'Mode', 'Midrange']
  losses = [mean_loss, median_loss, mode_loss, midrange_loss]
```

```
    # Plot
    plt.figure(figsize=(10, 6))
    plt.bar(measures, losses, color=['blue', 'green', 'orange', 'red'])
    plt.title('Loss (Max Absolute Deviation) for Different Central Measures')
    plt.ylabel('Max Absolute Deviation')
    plt.xlabel('Central Measure')
    plt.ylim(0, max(losses) + 5)

    # Show data points and central measures
    plt.scatter([0]*len(data), data, color='blue', label='Data points', marker='x')
    plt.scatter([1]*len(data), data, color='green', label='_nolegend_')
    plt.scatter([2]*len(data), data, color='orange', label='_nolegend_')
    plt.scatter([3]*len(data), data, color='red', label='_nolegend_')

    plt.legend()
    plt.show()

    # Output the losses for each measure
    print(f"Mean Loss: {mean_loss}")
    print(f"Median Loss: {median_loss}")
    print(f"Mode Loss: {mode_loss}")
    print(f"Midrange Loss: {midrange_loss}")

np.random.seed(11)  # For reproducibility
data = np.random.randint(1, 100, size=100)

mean, median, mode, midrange = calc(data)
mean_loss, median_loss, mode_loss, midrange_loss = cal_loss(data, mean, median, mode, midrange)
plotter(mean_loss, median_loss, mode_loss, midrange_loss)
```

Plot: Loss by using mean, median, mode, midrange on a Randomized Distribution

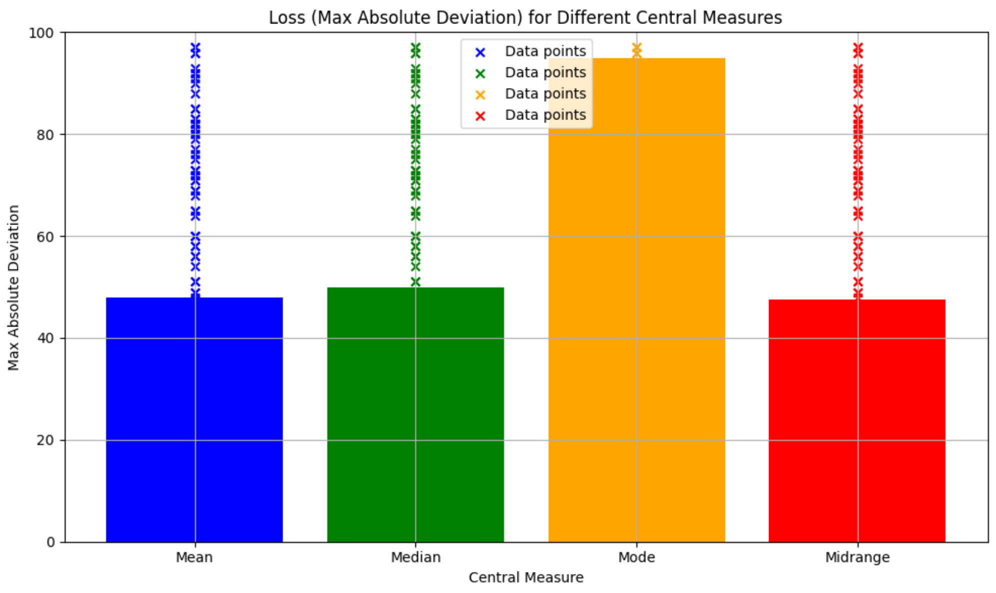

Figure 1: Loss with different central tendencies for a randomized simple distribution

This plot demonstrates the different values of the loss function for different values of m.
Mean Loss: 47.93
Median Loss: 50.0
Mode Loss: 95
Midrange Loss: 47.5

Plot: Loss by using mean, median, mode, midrange on a skewed Distribution

```
np.random.seed(11)  # For reproducibility
data_skewed = np.random.exponential(scale=10, size=100).astype(int)

mean, median, mode, midrange = calc(data_skewed)
mean_loss, median_loss, mode_loss, midrange_loss = cal_loss(data, mean, median, mode, midrange)
plotter(mean_loss, median_loss, mode_loss, midrange_loss)
```

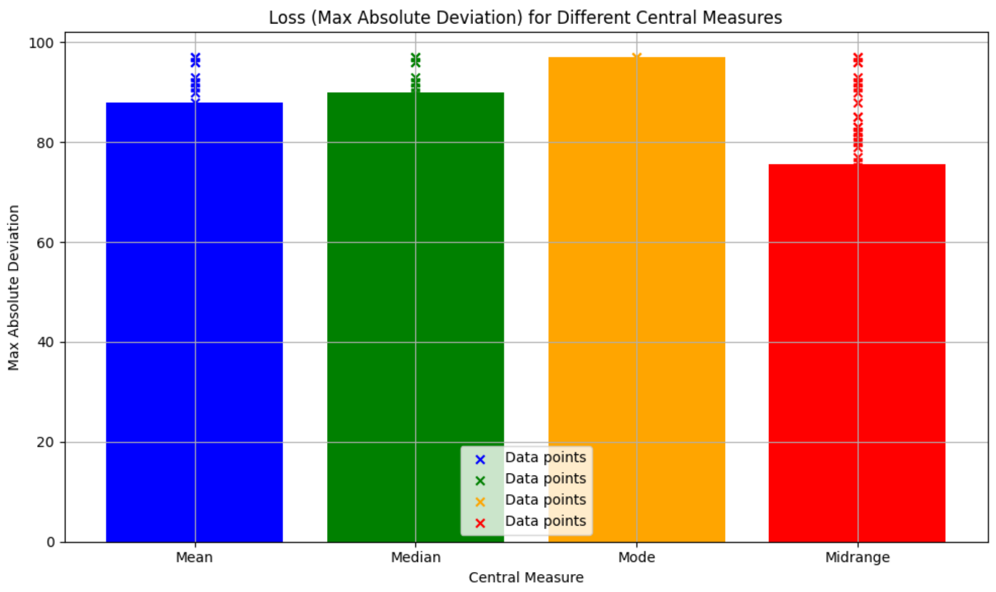

Figure 2: Loss with different central tendencies for a randomized skewed distribution

This code is for the plot which demonstrates the difference between losses for different values of m.
Mean Loss: 87.96000000000001
Median Loss: 90.0
Mode Loss: 97
Midrange Loss: 75.5

Both the above plots as well as the **Midrange Loss** readings showcase that the midrange minimizes the maximum absolute deviation.

c. Minimizing Absolute Deviation

We are given the absolute deviation loss function which is to be minimized. This loss fucntion can be represented as:

$$L(m) = \sum_{i=1}^{n} |y_i - m|$$

We can prove that median minimizes this loss function, in a few steps as belows:

1. Sort the data points: We will first sort the data as a sorted distribution is the pre-requisite of calculating the median, we will start of with it. Therefore, arrange the numbers such that $y_1 \leq y_2 \leq ... \leq y_n$.

2. Let m be any value in the sorted dataset. The loss can then be written as:

$$L(m) = \sum_{y_i < m} (m - y_i) + \sum_{y_i > m} (y_i - m)$$

3. Consider moving m slightly to the left (decreasing it). This will:
- Decrease the loss for all points to the right of m
- Increase the loss for all points to the left of m

4. The total change in the loss will be:

$$\frac{dL}{dm} = (\text{number of points} \geq m) - (\text{number of points} < m)$$

There is a change as the the derivative will be negative when there are more points to the right of m, and positive when there will be more points to the left.

For the loss to be minimized, the derivative needs to be zero, which can happen only when there are an equal number of points on each side of m.

5. This can happen when m is in the middle of the distribution, and that is possible when when m is precisely at the median of the data.

Hence, we have proved that the absolute deviation is minimized when m is the median of the data.

Problem 3: Piecewise Linear Regression via Feature Transformations

Collaborators: None.

Given is a single-variate dataset of the form $(x_1, y_1), \ldots, (x_n, y_n)$ where all values are scalars. Our task is to fit a piecewise linear model with two pieces, split at a known value λ.

a Equivalence of Constrained and Unconstrained Models

Given under is the piecewise linear model:

$$f(x_i) = \begin{cases} a_1 + s_1 x_i & \text{for } x_i < \lambda \\ a_2 + s_2 x_i & \text{for } x_i \geq \lambda \end{cases}$$

with the additional constraint that $a_1 + s_1\lambda = a_2 + s_2\lambda$. This constraint ensures that the two linear models meet at $x = \lambda$, making the prediction function smooth and continuous.

The unconstrained model is as

$$f(x_i) = \begin{cases} a_1 + s_1 x_i & \text{for } x_i < \lambda \\ a_1 + s_1\lambda - s_2\lambda + s_2 x_i & \text{for } x_i \geq \lambda \end{cases}$$

We can prove that both these models are equivaluent in a few steps:

1. The first piece of both the models represented by $(x_i < \lambda)$, is same and hence is equal.

2. From the constraint $a_1 + s_1\lambda = a_2 + s_2\lambda$:
We can express a_2 in Terms of a_1 and λ as:

$$a_2 = a_1 + s_1\lambda - s_2\lambda$$

3. Substitute this value of a_2 into the second piece of unconstrained model

$$f(x_i) = a_1 + s_1\lambda - s_2\lambda + s_2 x_i \text{ for } x_i \geq \lambda$$

$$f(x_i) = (a_2) + s_2 x_i \text{ for } x_i \geq \lambda$$

4. Combining the pieces of unconstrained model we get:

$$f(x_i) = \begin{cases} a_1 + s_1 x_i & \text{for } x_i < \lambda \\ (a_2) + s_2 x_i \text{ for } x_i \geq \lambda \end{cases}$$

This is identical to the constrained model and hence we have shown the equivalence between the constrained and unconstrained model.

(b) Fit the Model using Multiple Linear Regression

To fit the piecewise linear model using multiple linear regression, we need to perform a set of 3 steps.

Step 1: Transform the Input Data:

We will need to transform the input data from a $(x_1, y_1), \ldots, (x_n, y_n)$ form to a matrix form in order to use them for piecewise multiple regression.

Therefore, we need to define a new feature matrix $\mathbf{X}$ with multiple columns, whose values are derived from the original data points $(x_1, y_1), \ldots, (x_n, y_n)$ and a known value λ. The feature matrix van be defined as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & \max(0, x_1 - \lambda) \\ 1 & x_2 & \max(0, x_2 - \lambda) \\ \vdots & \vdots & \vdots \\ 1 & x_n & \max(0, x_n - \lambda) \end{bmatrix}$$

This matrix $\mathbf{X}$ has three important aspects:
1. the first column has 1s for to account for the intercept term and preserving linearity during matrix multiplications.
2. The second column has all x_i values
3. the third column has the positive difference between x_i and the known value λ, or a zero if the difference is negative.

Step 2. Use Multiple Regression Algorithm:

With the transformed data matrix $\mathbf{X}$, the problem now becomes a standard multiple linear regression. We can use a multiple regression algorithm to find the parameters $\boldsymbol{\beta}$ that minimize the squared loss:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

As, due to the data transformation, it is now a a simple multiple linear regression problem, the model form will be

$$f(x_i) = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i}$$

which can be solved to minimize the sqaured error, in order to find the optimized parameters

$$\beta = [\beta_0, \beta_1, \beta_2]^T$$

by using the given formula:

$$\beta = (X^T X)^{-1} X^T y$$

where,
X is the transformed data matrix
y is the vector of observed values $[y_1, y_2, \ldots, y_n]^T$.

Step 3. Extract Optimal Values:

From the optimal $\boldsymbol{\beta}$, we can extract the optimal values for a_1, s_1, s_2.

Let

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

Then, we can extract the following values:

- $a_1 = \beta_0$ (intercept)
- $s_1 = \beta_1$ (slope for $x < \lambda$)
- $s_2 = \beta_1 + \beta_2$ (slope for $x \geq \lambda$)

These values will ensure that the piecewise linear model fits the data optimally under the squared loss.

To confirm the correct working of our model we can verify by substituting the extracted values in given model functions.

For $x_i < \lambda$:

$$f(x_i) = \beta_0 + \beta_1 x_i + \beta_2 \cdot 0 = a_1 + s_1 x_i$$

For $x_i \geq \lambda$:

$$\begin{aligned} f(x_i) &= \beta_0 + \beta_1 x_i + \beta_2 (x_i - \lambda) \\ &= \beta_0 + \beta_1 \lambda + (\beta_1 + \beta_2)(x_i - \lambda) \\ &= a_1 + s_1 \lambda + s_2 (x_i - \lambda) \end{aligned}$$

This is identical to our unconstrained model from part (a) of the problem. Therefore, this proves that the approach is correct and fits the desired piecewise linear function.

(c) Python Implementation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

names = ['mpg', 'cylinders', 'displacement', 'horsepower',
         'weight', 'acceleration', 'model year', 'origin', 'car name']
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data',
                   header=None, delim_whitespace=True, names=names, na_values='?')
data = data.dropna()

X = data['horsepower'].values
y = data['mpg'].values
```

```
# Transform the input data into the feature matrix compatible for piecewise regression

lambda_value = 100
X_final = np.column_stack([
    np.ones(len(X)),  # Intercept - first columns
    X,  # Horsepower values
    np.maximum(0, X - lambda_value)  # Max(0, x - lambda) ])

# Compute beta* coefficients using the least squares

beta = np.linalg.inv(X_final.T @ X_final) @ X_final.T @ y
beta_0, beta_1, beta_2 = beta

# Piecewise linear function

def piecewise_linear(X, beta_0, beta_1, beta_2, lambda_value):
    return np.where(X < lambda_value,
                    beta_0 + beta_1 * X,  # For x < lambda
                    beta_0 + beta_1 * lambda_value + (beta_1 + beta_2) * (X - lambda_value))  # For x >

# Predict
y_pred = piecewise_linear(X, beta_0, beta_1, beta_2, lambda_value)

# Plotter
plt.figure(figsize=(10, 5))

plt.scatter(X, y, color=’blue’, marker=’x’, label=’Actual Data’, zorder=5)

plt.plot(np.sort(X), piecewise_linear(np.sort(X), beta_0, beta_1, beta_2, lambda_value),
         color=’red’, label=’Piecewise Linear Fit’, zorder=4)

plt.axvline(x=lambda_value, color=’green’, linestyle=’--’, label=f’ = {lambda_value}’)

plt.xlabel(’Horsepower’)
plt.ylabel(’MPG’)
plt.tight_layout()
plt.legend()
plt.grid(True)
```

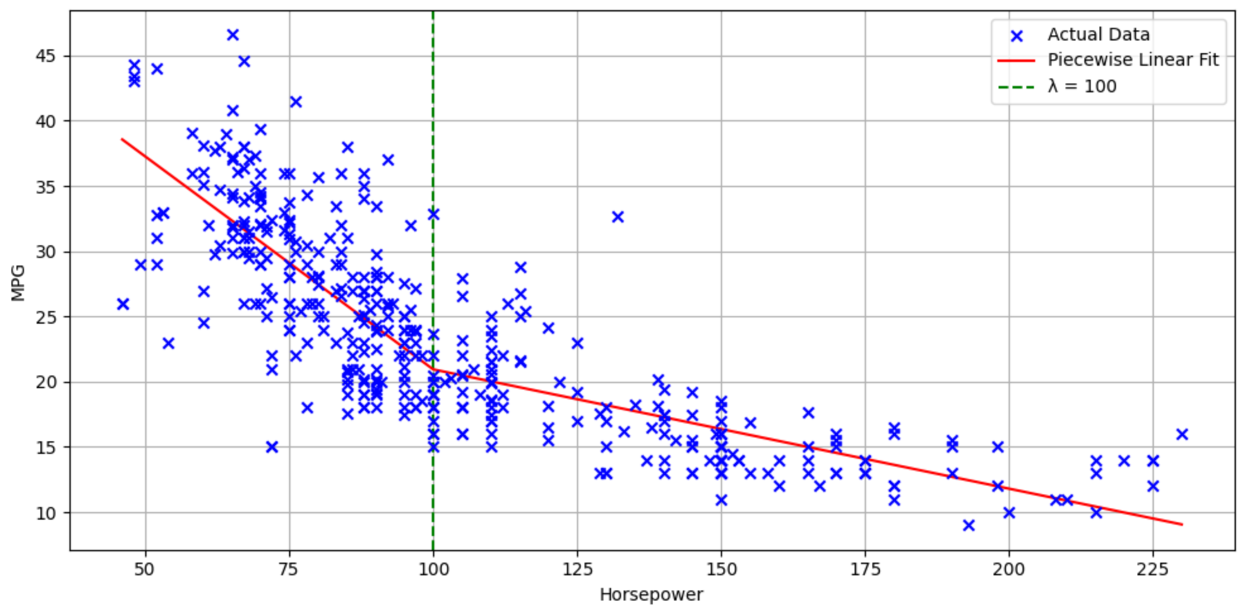

Figure 3: Piecewise Linear Regression Result

This figure verfies the correct implementation of piece wise linear regression and hence proves that the correctness of the algorithm.

Hence, demonstrated.

Problem 4: Thinking About Data Transformations

Collaborators: None.

a. Mean Centering

In this part of the problem, as the friend suggested to perform mean centering, therefore each entry in column i (except the first column of ones) is subtracted from the column mean $\bar{x}_i$.

This transforms the data mAtrix X to X' which is as follows:

$$X' = \begin{bmatrix} 1 & (x_{1,1} - \bar{x}_1) & (x_{1,2} - \bar{x}_2) & \cdots & (x_{1,d} - \bar{x}_d) \\ 1 & (x_{2,1} - \bar{x}_1) & (x_{2,2} - \bar{x}_2) & \cdots & (x_{2,d} - \bar{x}_d) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_{n,1} - \bar{x}_1) & (x_{n,2} - \bar{x}_2) & \cdots & (x_{n,d} - \bar{x}_d) \end{bmatrix}$$

where, $\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{j,i}$.

We know that the multiple linear regression model finds β that minimizes $\|y - X\beta\|_2^2$. With mean-centering, we will be minimizing $\|y - X'\beta'\|_2^2$.

But, both these predictions are equal if: $\beta_i' = \beta_i$ for $i = 1, ..., d$

and we can show that

$$X'\beta' = X\beta + \begin{bmatrix} \beta_0 - \sum_{i=1}^{d} \beta_i \bar{x}_i \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Hence, this shows that original and mean centred data matrices will showcase same results.
This is due to the fact that the mean centering operation does not change the total sum of squares.
It only shifts the individual values and the model adapts to this shift just by adjusting its intercept.
And the rest is similar, and no change in the overall predictions and loss.

b. Normalizing to Unit Standard Deviation

In this part of the problem, we were suggested to perform normalization to unit standard deviation, therefore each entry in column i (except the first column of ones) is divided by its standard deviation σ_i.

This normalizes the data matrix X to X" which is as follows:

$$X'' = \begin{bmatrix} 1 & \frac{x_{1,1}}{\sigma_1} & \frac{x_{1,2}}{\sigma_2} & \cdots & \frac{x_{1,d}}{\sigma_d} \\ 1 & \frac{x_{2,1}}{\sigma_1} & \frac{x_{2,2}}{\sigma_2} & \cdots & \frac{x_{2,d}}{\sigma_d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \frac{x_{n,1}}{\sigma_1} & \frac{x_{n,2}}{\sigma_2} & \cdots & \frac{x_{n,d}}{\sigma_d} \end{bmatrix}$$

where $\sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_{j,i} - \bar{x}_i)^2}$.

We know that the multiple linear regression model finds β that minimizes $\|y - X\beta\|_2^2$. With normalization, we will be minimizing $\|y - X''\beta''\|_2^2$

But, both these predictions are equal if: $\beta_i' = \beta_i$ for $i = 1, ..., d$ and we can show that

$$X''\beta'' = X\tilde{\beta}$$

where, $\tilde{\beta}_i = \frac{\beta_i''}{\sigma_i}$ for $i > 0$, and $\tilde{\beta}_0 = \beta_0''$.

Hence this shows that original and normalized data matrices will showcase same results.
This is due to the fact that normalization scales the data columns, and the model can adapt to this scaling by adjusting its coefficients proportionally, resulting in the same predictions and loss. Therefore, there are no accuracy-based improvements in the overall model.

(c) Impact on ℓ_1 and ℓ_∞ Loss

The ℓ_1 and ℓ_∞ Loss functions are given below:
ℓ_1 loss:

$$L_1(\beta) = \sum_{i=1}^{n} |y_i - (X\beta)_i|$$

ℓ_∞ loss:

$$L_\infty(\beta) = \max_i |y_i - (X\beta)_i|$$

For ℓ_1 Loss:
ℓ_1 loss minimizes the sum of absolute errors.
While mean centering or normalizing can change the distribution of residuals, it does not fundamentally change the minimization problem because the absolute differences remain proportional.

For ℓ_∞ Loss:
ℓ_∞ loss minimizes the maximum absolute error.
Again, the transformations do not affect the maximum error because scaling or shifting all errors by the same proportion does not change the maximum value.
But there is a chance that normalization may result in distinct solutions, because of modifications to the optimization process.

Therefore, theoretically mean centering and normalization do not improve the model loss, as the relationships between residuals and the data remain unchanged, though when utilizing specific optimization algorithms, there may be minor changes for ℓ_1 and ℓ_∞ loss.

Problem 5: Student Question: Slightly More Efficient One-hot Encoding

Collaborators: None.

Let $\mathbf{X}$ be our one-hot encoded data matrix, which also includes an intercept column of ones. Let $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_k$ be the k columns corresponding to the one-hot encoded categorical variable, where, every row in $\mathbf{X}$, will have only a single '1' (one) value and the rest will be 0 (zero).

This matrix, $\mathbf{X}$ is linear and can be represented as: $y = \mathbf{X}\boldsymbol{\beta}$ where $\boldsymbol{\beta}^* = [\beta_0, \beta_1, ..., \beta_j, ..., \beta_k, ...]^T$ are the coefficients. Here, β_0 is the column that represents all 1s and $\beta_1, ..., \beta_k$ correspond to the one-hot encoded columns.

The loss function for this model is given by:

$$\text{Loss} = \|y - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_{c(i)})^2$$

where $c(i)$ indicates the category for the i-th row.

Now as informed, we can remove any one column out of the k columns without losing information. This column is still uniquely identified as it has all zeroes. We decide to remove the j-th column and now the new data matrix is represented by **X'** .

And the the new coefficient vector $\boldsymbol{\beta'}$ for the new reduced matrix $\mathbf{X}'$ can be defined as:

$$\boldsymbol{\beta'} = [\beta'_1, \beta'_2, \ldots, \beta'_{j-1}, \beta'_{j+1}, \ldots, \beta'_k]^T$$

We can also relate the new coefficients β'_i to the original coefficients as:

$$\beta'_i = \beta_i - \beta_j \quad \text{for } i \neq j$$

For this reduced model, the prediction for categories $i \neq j$, which is when ($i < j$ or $i > j$), the prediction will be:

$$y'_i = \beta'_i + \beta_j = \beta_i$$

Thus, the predictions remain unchanged for all categories other than j.

For the Missing Category j, the prediction is simply:

$$y'_j = \beta_j$$

Now, We need to PROVE that the loss function with the reduced matrix $\mathbf{X}'$ gives the same result as with the original matrix $\mathbf{X}$.
The Original Loss Function:

$$\text{Loss} = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_{c(i)})^2$$

The Reduced Loss Function:

$$\text{Loss}' = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta'_{c(i)} - \beta_j \right)^2$$

Since we have $\beta'_{c(i)} = \beta_{c(i)} - \beta_j$, we substitute this into the reduced loss function:

$$\text{Loss}' = \sum_{i=1}^{n} \left(y_i - \beta_0 - (\beta_{c(i)} - \beta_j) - \beta_j \right)^2$$

Simplifying further as β_j gets cancelled out:

$$\text{Loss}' = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_{c(i)})^2 = \text{Loss}$$

This shows that the loss remains unchanged when we remove one column from the one-hot encoded matrix. The removal of any one column from the one-hot encoded matrix and the corresponding adjustment to the coefficients do not change the model's predictions or the loss function. This proves that we can safely reduce the number of columns in the one-hot encoding without losing any information.

This also proves that we can always encode k categories with k 1 columns without introducing unintentional linear relationship. This is because we can represent the k-th category implicitly, using the absence of any of the $k1$ categories.