EN3150

PATTERN RECOGNITION

ASSIGNMENT 01

LEARNING FROM DATA AND RELATED CHALLENGES AND LINEAR MODELS FOR REGRESSION

03.09.2024

AMARASURIYA G.N.

210036D

1) Data Pre-processing

Feature 1: Max-Abs Scaling

Given the sparsity and presence of significant outliers in Feature 1, Max-Abs scaling is appropriate. This method will preserve the zero values and the general structure of the feature. This brings the outliers into a comparable range. It also prevents the skewing effect that standard scaling or min-max scaling might introduce due to the outliers.

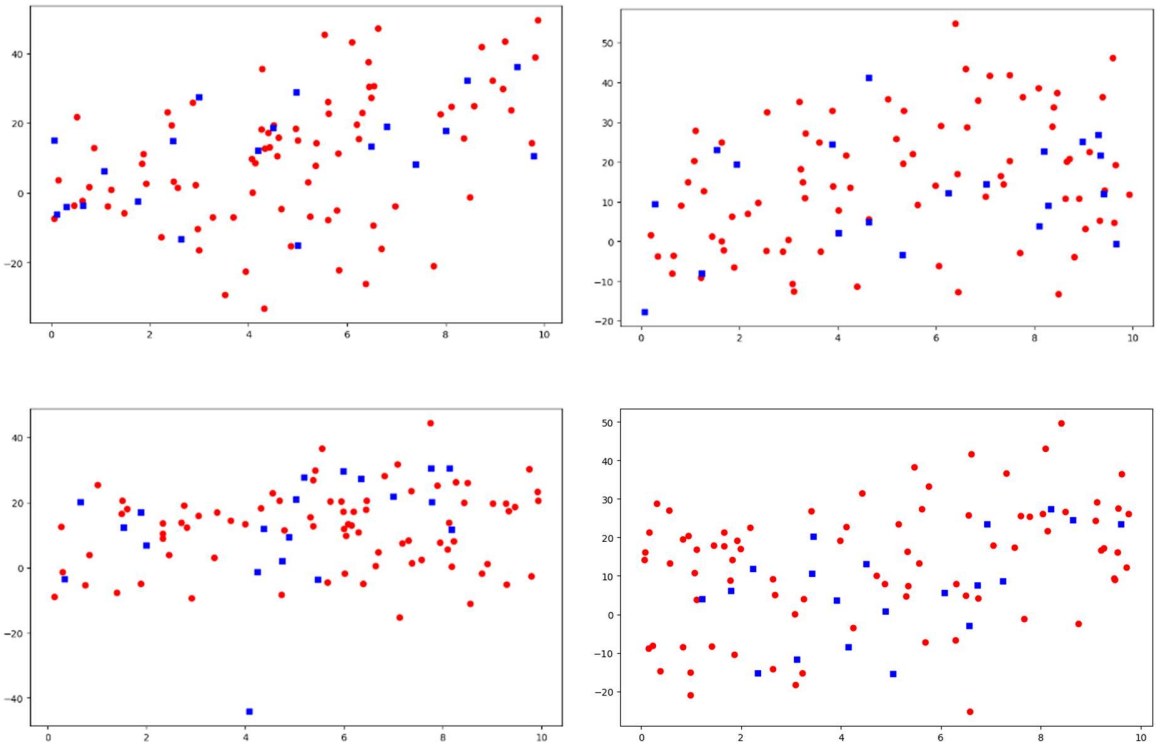
Feature 2: Standard Scaling

Feature 2 has a wide range and has both positive and negative values without being sparse. Standard scaling is suitable here as it will normalize the feature, bringing it to a common scale without affecting the distribution structure. This method is less sensitive to outliers compared to min-max scaling. Since Feature 2 has more variability, standard scaling will handle it effectively.

2) Learning from data

1. Done in Q2.ipynb

2. Simulated in Q2.ipynb with screenshots of trials attached below



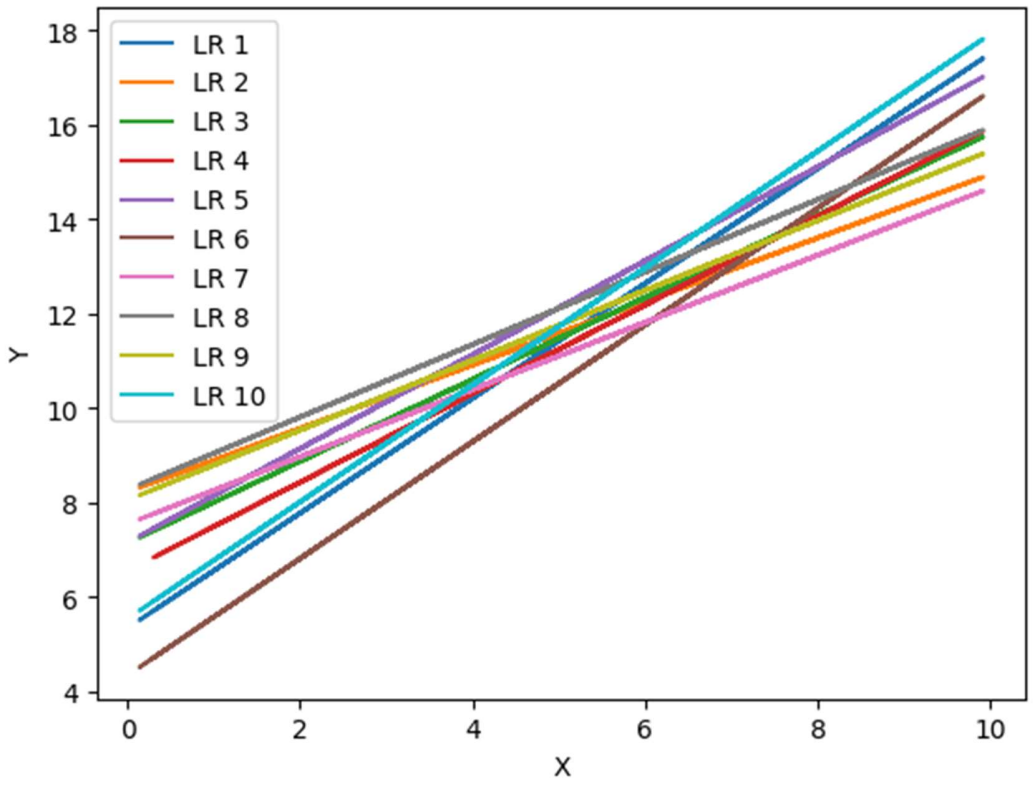
The code generates unique scatter plots for each run. Each displays different positions of training and testing data points. These plots reflect changes in data distribution between sets. Despite variations in specific data points, the overall trend should show some similarity.

This variation occurs because of the random processes involved namely.

Random Data Generation: The use of np.random.rand and np.random.normal introduces randomness in each run. It leads to different X and Y values every time the code is executed. The split is determined by the random_state parameter, which is set to a randomly generated integer (r = np.random.randint(104)). Since this integer changes with each execution, the train_test_split function will create different training and testing sets every time the code is run.

Train-Test Split Shuffling: The train_test_split function randomly mixes the data before splitting them to training and test sets. This randomness helps to avoid the model from overfitting to specific data patterns, thus results in a more generalizable modelling.

3.



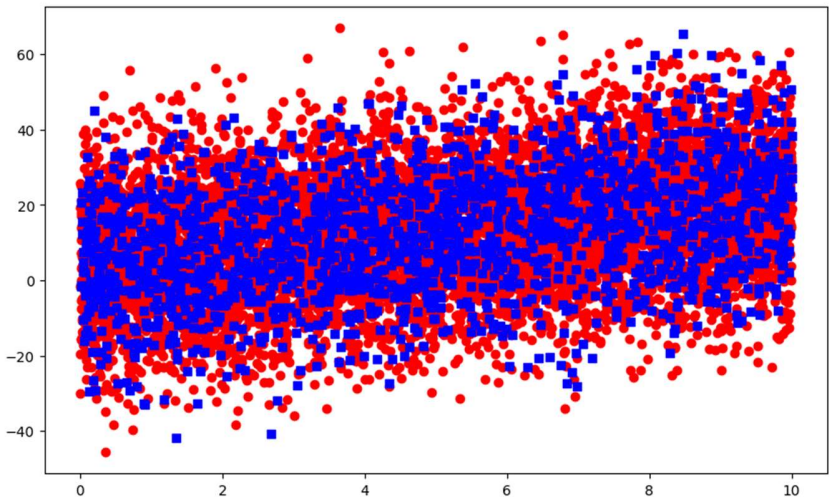
The code runs a linear regression model training process 10 times, each time with a slightly different model.

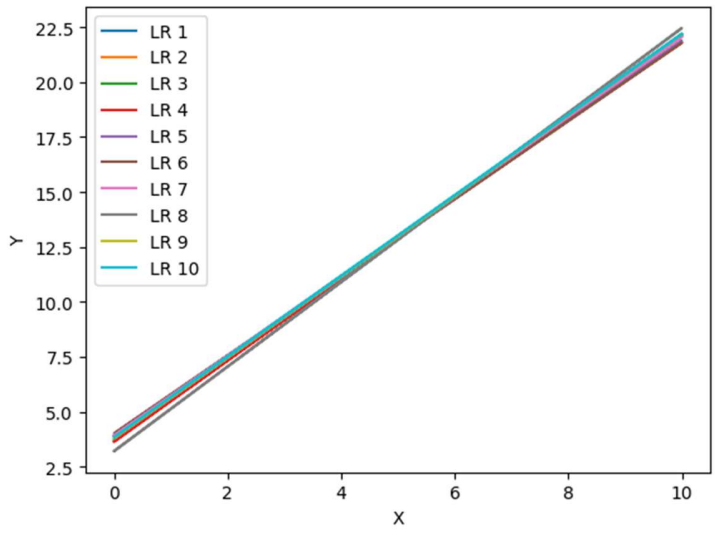
The reasons for the difference from one instance to other instance can be

- **Random Train-Test Split:** The train_test_split function shuffles data into training and testing sets using a random integer seed. This results in different training data permutations, affecting model parameters. In turn, resulting in slightly different models despite the overall dataset remaining unchanged.
- **Variability in the Error Term (Epsilon):** The model being fit is Y = 2X + 3 + epsilon, which represents random noise or error. The epsilon term, generated using a normal distribution with zero mean, causes minor differences in the fitted model affecting the overall trend of the data.

These variations are small, but they highlight the inherent randomness in linear regression, like when dealing with noisy data.

4. Increasing the number of data samples to 10,000 (n_samples = 10000 in listing 1) and repeating the task 3. Can observe data scatter and model as follows.





Compared to the previous we can observe

- **Reduced Variability:** The model's variability and overfitting risk decrease with 10,000 data samples. This enhances generalization to unseen data and reduces the likelihood of overfitting.
- **Consistency in Linear Regression Lines:** By using 10,000 samples, the linear regression lines in each iteration become more consistent. They are closely aligned, reducing variation and almost overlapping.
- **Improved Model Accuracy and Confidence:** A larger dataset enhances the accuracy and confidence of a model, as it better captures the relationship between features and the target variable.

Reason for the different behavior compared to 100 data samples:

- **Increase with Large Numbers:** As the sample size increases, the sample mean and variance converge towards the true population mean and variance.
- **Diminished influence of outliers and variations:** With a larger dataset, these outliers and variations are diluted, leading to more robust and consistent models. This also contributes to improved generalization, as the model is less likely to overfit to noise in the data.

3) Linear regression on real world data

1. Done in Q3.ipynb

2. Number of independent variables: 33

Number of dependent variables: 2

3. Linear regression is suitable when target variable is numerical/continuous. If there is categorical, linear regression is not suitable.

From the 33 variables there are 4 categorical variables – SubjectID, Gender, Age, Ethnicity

Linear regression cannot directly handle categorical variables. Therefore, we need to convert them into numerical representations using encoding techniques.

E.g. 1) **One-Hot Encoding:** This method creates a new binary column for each category in the categorical variable.

2) **Label Encoding:** This method converts each category to an integer value.

4. Using **Q3.ipynb block 5 code** can observe that “Distance” has 2 NaN values and no NaN values in target variables.

Normally if there lot of NaNs we should replace or drop. But using X = X.dropna() and y = y.dropna() in this case is not ideal. It would drop entire rows across all features and targets. This might result in losing more data than necessary.

A better approach is to drop only the two rows corresponding to the NaN values in the "Distance" feature. Done in **Q3.ipynb block 6.**

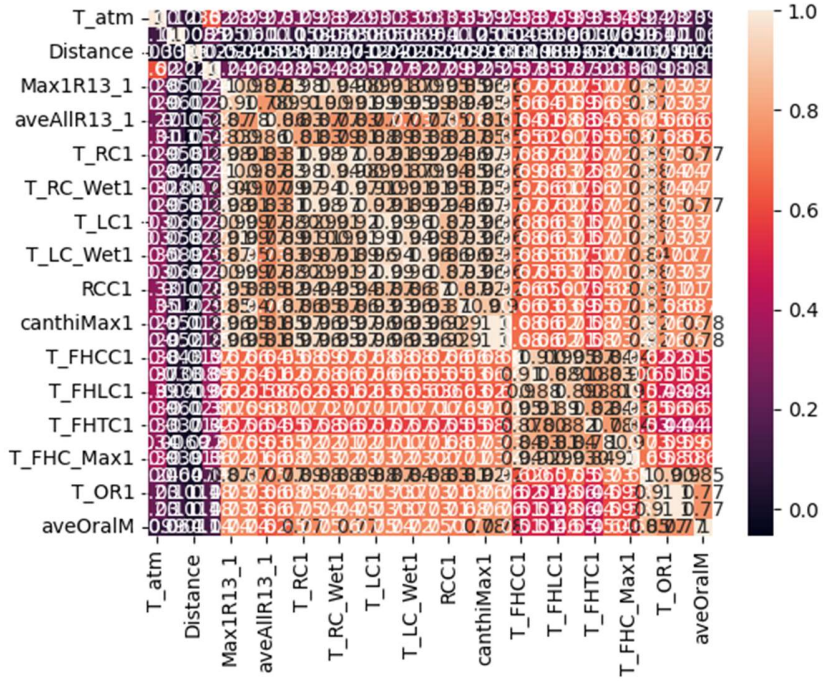
5. Q3.ipynb blocks 7-10 code

"aveOralM" is assigned as the dependent feature.

To select the best features of X can test the correlation of features in order of how strongly they are correlated with "aveOralM" using the correlation matrix.

```
corr_matrix = combined_dropped.corr(numeric_only=True)
sns.heatmap(corr_matrix, annot=True)
plt.show()
```

```
target_corr = corr_matrix['aveOralM'].drop('aveOralM')
sorted_corr = target_corr.abs().sort_values(ascending=False)
print(sorted_corr)
```




```
T_Max1          0.849657
canthiMax1      0.784096
canthi4Max1     0.783763
T_RC_Max1       0.769463
T_RC1           0.768730
T_OR1           0.767345
T_OR_Max1       0.766911
T_RC_Wet1       0.760134
T_LC_Max1       0.750527
T_LC1           0.750390
T_RC_Dry1       0.748566
T_LC_Dry1       0.743656
Max1R13_1       0.741902
Max1L13_1       0.735644
T_LC_Wet1       0.720383
RCC1            0.714193
LCC1            0.696752
aveAllL13_1     0.645084
T_FH_Max1       0.641727
aveAllR13_1     0.620874
T_FHC_Max1      0.595470
T_FHBC1         0.552362
T_FHRC1         0.514298
T_FHCC1         0.513335
T_FHLC1         0.463579
T_FHTC1         0.454889
T_offset1       0.139041
T_atm           0.098663
Humidity        0.063595
Distance        0.011181
Name: aveOralM, dtype: float64
```

Selected independent features: 'Age', 'T_Max1', 'canthiMax1', 'canthi4Max1', 'T_RC_Max1'

Dependent and Independent features in Y1 and X1 respectively.

6. Q3.ipynb block 11 code

The dataset split into training and testing sets, with 80% of the data points used for training and 20% for testing using train_test_split.

Added np.random.randint to introduce randomness in the random_state.

7. Q3.ipynb blocks 12-14 code

Linear Regression Model from sklearn trained using training variables.

The intercept gives the baseline prediction when all independent variables are zero.

Intercept: 4.9151558

The coefficients show the effect of each independent variable on the dependent variable.

```
coefficients = pd.Series(model.coef_[0], index=X1.columns)
```

Estimated Coefficients for Independent Variables:

Age = 0.008188
T_Max1 = 0.834347
canthiMax1 = -0.413000
canthi4Max1 0.305199
T_RC_Max1 = 0.164414

8. Q3.ipynb block 15 code

```
most_significant_variable = coefficients.abs().idxmax()
```

The independent variable that contributes the most to the dependent feature is T_Max1.

9. Q3.ipynb blocks 16-17 code

Linear regression model trained on given features and coefficients obtained as in the code block. Added np.random.randint to introduce randomness in the random_state.

```
Intercept:
 [7.03315036]
Estimated Coefficients for Independent Variables:
T_OR1        -0.574011
T_OR_Max1     1.126371
T_FHC_Max1   -0.067109
T_FH_Max1     0.353906
dtype: float64
```

10. Q3.ipynb blocks 18-20 code

Computed using standard equations. Obtained the following values.

Residual sum of squares (RSS)	74.491067
Residual Standard Error (RSE)	0.303443
Mean Squared Error (MSE)	0.09528609
R^2 statistic	0.656808

The statistics for each feature were obtained as follows

	Standard Error	**t-statistic**	**p-value**
T_OR1	0.000357931297	-30.340337525570	0.000542278288
T_OR_Max1	0.000358201386	59.513818871974	0.000141107627
T_FHC_Max1	0.000326562220	-3.7136478137644	0.032734812719
T_FH_Max1	0.000394508116	17.8180276250273	0.001567489891

11. A high p-value doesn't necessarily mean a feature is unimportant. It might be correlated with other features that are important. Removing it could lead to losing valuable information.

(The code was run for second time for accuracy testing)

4) Linear regression on real world data

1. Table 1: SSE and TSS of linear regression models.

	Model A	Model B
SSE= $\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$	9	2
TSS= $\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2$	90	10
Number of data samples (N)	10000	10000

2. From the lecture note -> Residual Sum of Squares (RSS) = Sum of Squared Error (SSE)

Calculating Residual Standard Error (RSE) for models A and B using

$$\text{RSE} = \sqrt{\frac{\text{SSE}}{\text{N} - \text{d} - 1}}$$

For Model A:

- SSE=9
- N=10000
- k=2 (predictors: x_1, x_2)

$$\text{RSE}_A = \sqrt{\frac{9}{10000 - 2 - 1}} \approx 0.03$$

For Model B:

- SSE=2
- N=10000
- k=4 (predictors: x_1, x_2, x_3, x_4)

$$\text{RSE}_B = \sqrt{\frac{2}{10000 - 4 - 1}} \approx 0.01415$$

∴ Based on RSE, Model B performs better because it has a lower RSE (0.01415) compared to Model A (0.03).

3. From the lecture.

Calculating R-squared (R^2) for models A and B using (Using RSS=SSE)

$$R^2 = 1 - \frac{SSE}{TSS}$$

For Model A:

- SSE=9
- TSS=90

$$R_A^2 = 1 - \frac{9}{90} = 0.9$$

For Model B:

- SSE=2
- TSS=10

$$R_A^2 = 1 - \frac{2}{10} = 0.8$$

∴ Based on R^2 value, Model A performs better because it has a higher R^2 (0.9) compared to Model B (0.8).

4. Both the RSE and R^2 help in understanding model performance.

- **RSE** provides an overall sense of how far on average the observed outcomes are from the regression line. RSE gives a better idea about how well our model fits out data without being too complex.
- **R^2** is the relative measure of the fit of the model, indicates how much variance explained by this model.

∴ **RSE might be considered fairer** because it directly accounts for the complexity of the model. Whereas R^2 might not adequately penalize for overfitting when more predictors are included.

5) Linear Regression impact on outliers

1.

$$L_1(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{r_i^2}{a^2 + r_i^2}\right) = \frac{1}{N}\sum_{i=1}^{N}\left(L_{1,i}\right).$$

$$L_2(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(1 - \exp\left(\frac{-2|r_i|}{a}\right)\right) = \frac{1}{N}\sum_{i=1}^{N}\left(L_{2,i}\right).$$

2.

The loss function measures the difference between predicted and observed values for each data point. The main objective of training a model is to minimize this loss function.

Standard Least Squares (LS) Loss Function:

$$L_{LS}(w) \approx \frac{1}{N}\sum_{i=1}^{N} r_i^2$$

This function is highly sensitive to outliers because the squared term can amplify large residuals r_i

Modified Loss Functions L1(w) and L2(w):

For L1(w):

From the graph - L1(w) for small residuals ri approaches 1, and for large residuals, it rapidly decreases to 0.

When **a**→0 $L_{LS,1}(w) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{r_i^2}{r_i^2} = \frac{1}{N}\sum 1 = 1$

For small **a**, the loss function becomes almost insensitive to outliers, effectively reducing their impact on the overall loss.

In comparison to standard LS, L1(w) provides a better approach by disregarding the influence of outliers, making the model less sensitive to extreme values.

For L2(w):

The graph shows - L2(w) also reduces the influence of large residuals, but in a smoother manner compared to L1(w).

When **a**→0 $L_{ls,2}(w) \approx \frac{1}{N}\sum_{i=1}^{N}(1-0) = \frac{1}{N}\sum 1 = 1$

The function becomes nearly constant for large residuals. Once a residual passes a certain threshold, it has minimal effect on the loss.

Compared to standard LS, L2(w) allows for some contribution from outliers but significantly reduces their impact.

Conclusion: As **a** approaches 0, both modified loss functions L1(w) and L2(w) become increasingly insensitive to outliers, unlike the standard LS function. Both functions aim to minimize the influence of large residuals by limiting their impact on the overall loss function. They ensure each residual contributes a maximum value (around 1) to the loss. This prevents any single large residual from dominating the overall loss.

3. To minimize the influence of data points where the residual $|r_i| \geq 40$, we can use the above analysis and the graph to choose appropriate values of the hyper-parameter a and the most suitable loss function.

Analyzing the Graph:

- L1(w) and L2(w) both as **"a"** decrease, influence of large diminishes significantly.
- When **a**=2.5, both L1(w) and L2(w) are almost constant and near zero for $|r_i| \geq 40$. This implies that the influence of residuals beyond this threshold is nearly eliminated.
- When **a**=25, there is still some influence for residuals around $|r_i| \geq 40$.
- When **a**=100, the influence of large residuals increases significantly.

Function Choice:

- L1(w) sharply reduces the influence of large residuals, making it a good choice to discard the influence of outliers.
- L2(w) offers a smoother reduction, still minimizing the impact of outliers but not as aggressively as L1(w).

Conclusion: To minimize the influence of data points with $|r_i| \geq 40|$, I would choose L1(w) with **a**=2.5 would nearly eliminate the influence of such outliers based on my above analysis.

References

- J. T. Barron, "A general and adaptive robust loss function," *arXiv (Cornell University)*, Jan. 2017, doi: 10.48550/arxiv.1701.03077.
- I. Selesnick, "Introduction to sparsity in signal processing," *NYU-Poly*, Nov. 2012.

THE END