University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 5 (Intake 2021)

EN3150 - Pattern Recognition

Assignment 01
Learning from data and related challenges and linear models for regression

Sehara G.M.M - 210583B

This report is submitted as a partial fulfillment for the module EN3150 - Pattern Recognition, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

Contents

1 Datapre-processing

1.1 Feature 1

The majority of the values in this feature are zero, with only a few non-zero values, making it a sparse feature. Therefore, our scaling approach should preserve this sparsity.

- **Standard Scaling**: This method might assign non-zero values to originally zero values in the dataset, making it unsuitable.
- **Min-Max Scaling**: Similar to standard scaling, this method can also assign non-zero values to originally zero values, so it is also unsuitable.
- **Max-Abs Scaling**: This method maintains the sparsity of the data while reducing its range.

Considering these factors, Max-Abs scaling is the most appropriate method for scaling this feature.

1.2 Feature 2

This feature appears to follow a normal distribution, so our scaling method should maintain the normality of the data.

- **Standard Scaling**: This method centers the data around zero with a unit standard deviation, preserving the normal distribution of the feature. Therefore, it is a suitable choice.
- **Min-Max Scaling**: While this method scales the data to a specific range (typically 0 to 1), it does not necessarily preserve the normality of the data, making it less suitable in this case.
- **Max-Abs Scaling**: Although this method retains sparsity, it does not necessarily maintain the normal distribution of the feature. It is better suited for sparse features and is not ideal for this feature.

Considering these factors, Standard Scaling is the most appropriate method for scaling this feature.

2 Learning from data

2.1 Random Split

The `train_test_split` function divides the dataset into two folds. This process is random, and the randomness is determined by the `random_state` argument. In the first line of Listing 2, a random integer less than 104 is generated, which is then used as the `random_state`. This is why running the code twice produces two different plots.

```
Residual Sum of Squares (RSS): 77.17661510648327
Residual Standard Error (RSE): 0.3088649324302553
Mean Squared Error (MSE): 0.09481156646988116
R-squared (R2) statistic: 0.6528397310358398

Standard Errors for each feature:
T_OR1: 0.8694665336716999
T_OR_Max1: 0.8672245002334331
T_FHC_Max1: 0.04429270282049222
T_FH_Max1: 0.048964051526189434

T-statistics for each feature:
T_OR1: 0.1763447758368738
T_OR_Max1: 0.48111454588726127
T_FHC_Max1: -2.049524154302016
T_FH_Max1: 7.545180752626417

P-values for each feature:
T_OR1: 0.860067221055746
T_OR_Max1: 0.630565185099945
T_FHC_Max1: 0.04073285925720997
T_FH_Max1: 1.216299743510608e-13
```

Figure 1: First Output

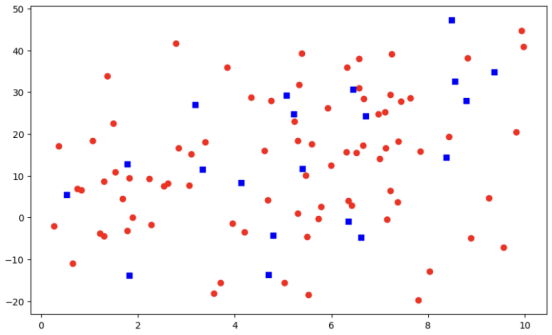

Figure 2: Second Output

2.2 Linear Regression

In Listing 2, ten models were trained using ten different randomly split training datasets. Because the model parameters depend on the training data, the parameters of the models vary. The graph below shows ten different lines, each representing a model trained on a distinct dataset split.

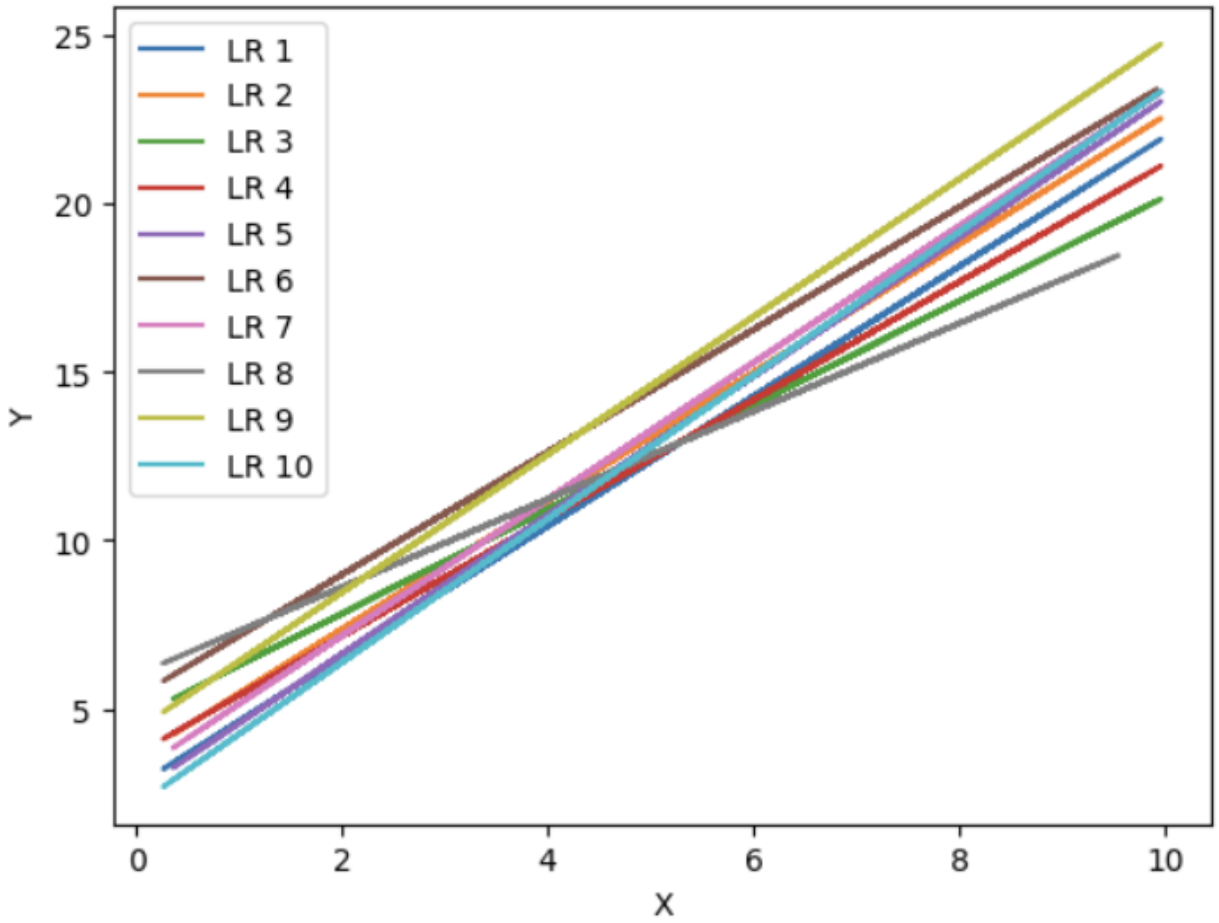

Figure 3: Lines given by different splits of same data set

2.3 Increasing the sample size

When the number of samples in a dataset is small, models tend to overfit to that dataset. However, in this code, the number of samples was increased, reducing overfitting and improving the models' ability to generalize across the dataset. As a result, even though random datasets were used as in the previous step, the models did not vary significantly, and the lines almost overlapped. The observed lines closely align with $y = 2x + 3$, which is the original model from which the data was generated.

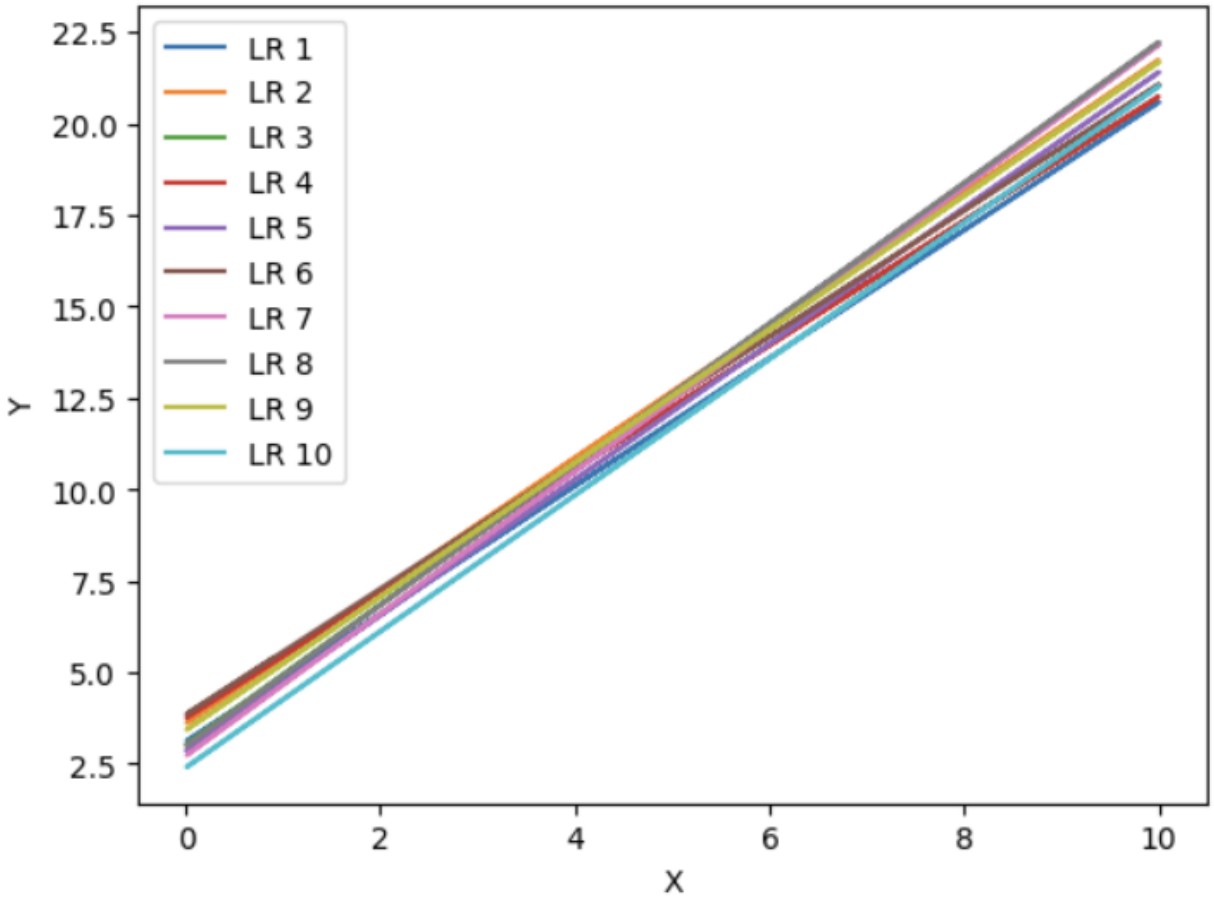

Figure 4: All the lines are almost the same with a higher number of samples

3 Linear Regression on Real World Data

3.2 Number of dependent and independent variables

- **Dependent Variables** - 2
- **Independent Variables** - 33

3.3 Is it possible to apply linear regression?

No
The dataset contains some categorical features, which cannot be directly used in linear regression. To perform linear regression, these categorical features must be encoded. You can use either one-hot encoding or label encoding, each of which has its own advantages and disadvantages.

3.4 Removing NaN/missing values

No
The provided method is incorrect because it independently drops features and dependent variables, which can lead to a mismatch in the dataset columns. Instead, X and Y should be concatenated, and the `dropna()` function should be applied to the combined data. Afterward, X and Y can be separated again.

Listing 1: Correct approach to handle missing values in X and Y

```
import pandas as pd

# Concatenate X and y
df = pd.concat([X, y], axis=1)

# Drop rows with missing values
df = df.dropna()

# Separate X and y again
X = df.drop(columns=y.columns)
y = df[y.columns]
```

3.5 Selecting features

Listing 2: Selected features

```
y_sel = df['aveOralM']

# Selecting the independent features
X_sel = df[['Age', 'Humidity', 'T_Max1', 'T_OR1', 'T_atm']]
```

3.6 Splitting data points

Listing 3: Splitting Data

```
X_train , X_test , Y_train , Y_test = train_test_split (X_sel, y_sel ,
test_size =0.2 , random_state = np . random . randint (104) )
```

3.7 Linear Regression Model

Listing 4: Linear Regression Model

```
df_encoded = pd.get_dummies(X, columns=['Age'], drop_first=True)
X_sel = df_encoded[['Humidity','T_Max1', 'T_OR1', 'T_atm',] + [col for col
in df_encoded.columns if col.startswith('Age_')]]
X_train , X_test , Y_train , Y_test = train_test_split (X_sel, y_sel ,
test_size =0.2 , random_state = np . random . randint(104) )
model = LinearRegression ()
model . fit ( X_train , Y_train )
coefficients = model.coef_
```

```
Residual Sum of Squares (RSS): 77.17661510648327
Residual Standard Error (RSE): 0.3088649324302553
Mean Squared Error (MSE): 0.09481156646988116
R-squared (R2) statistic: 0.652839731035839 8

Standard Errors for each feature:
T_OR1: 0.8694665336716999
T_OR_Max1: 0.8672245002334331
T_FHC_Max1: 0.04429270282049222
T_FH_Max1: 0.048964051526189434

T-statistics for each feature:
T_OR1: 0.1763447758368738
T_OR_Max1: 0.48111454588726127
T_FHC_Max1: -2.049524154302016
T_FH_Max1: 7.545180752626417

P-values for each feature:
T_OR1: 0.860067221055746
T_OR_Max1: 0.630565185099945
T_FHC_Max1: 0.04073285925720997
T_FH_Max1: 1.216299743510608e-13
```

Figure 5: Model coefficients

- Model Intercept = 5.3542183

3.8 Independent variable with the highest contribution to the dependent variable

The feature with the highest coefficient is T-Max1 (0.902573). Therefore T-Max feature is the independent variable with the highest contribution

3.9 By selecting 'T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1' features as independent features

Listing 5: Model for selected features

```
X_sel = df[['T_OR1', 'T_OR_Max1', 'T_FHC_Max1', 'T_FH_Max1']]
y_sel = df['aveOralM']

#split into training and testing data
X_train , X_test , Y_train , Y_test = train_test_split (X_sel, y_sel ,
test_size =0.2 , random_state = np . random . randint(104) )

model = LinearRegression ()
model . fit ( X_train , Y_train )
```

```
Residual Sum of Squares (RSS): 77.17661510648327
Residual Standard Error (RSE): 0.3088649324302553
Mean Squared Error (MSE): 0.09481156646988116
R-squared (R2) statistic: 0.6528397310358398

Standard Errors for each feature:
T_OR1: 0.8694665336716999
T_OR_Max1: 0.8672245002334331
T_FHC_Max1: 0.04429270282049222
T_FH_Max1: 0.048964051526189434

T-statistics for each feature:
T_OR1: 0.1763447758368738
T_OR_Max1: 0.48111454588726127
T_FHC_Max1: -2.049524154302016
T_FH_Max1: 7.545180752626417

P-values for each feature:
T_OR1: 0.860067221055746
T_OR_Max1: 0.630565185099945
T_FHC_Max1: 0.04073285925720997
T_FH_Max1: 1.216299743510608e-13
```

Figure 6: Model Coefficients

- Model Intercept = 7.326129

3.10 Calculating the parameters

Listing 6: Parameter Calculation

```
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score

# Add a constant (intercept) to the model
X_train_with_const = sm.add_constant(X_train)

# Fit the model using statsmodels
model_sm = sm.OLS(Y_train, X_train_with_const).fit()

# Predictions on the training set
y_train_pred = model_sm.predict(X_train_with_const)

# Residuals (actual - predicted)
residuals = Y_train - y_train_pred

# Calculate Residual Sum of Squares (RSS)
RSS = np.sum(residuals ** 2)

# Calculate Residual Standard Error (RSE)
N = len(Y_train)  # Number of samples
d = X_train.shape[1]  # Number of independent features
RSE = np.sqrt(RSS / (N - d - 1))

# Calculate Mean Squared Error (MSE)
MSE = mean_squared_error(Y_train, y_train_pred)

# Calculate R-squared (R2) statistic
R2 = r2_score(Y_train, y_train_pred)

# Get standard errors, t-statistics, and p-values for each feature
```

```
standard_errors = model_sm.bse
t_statistics = model_sm.tvalues
p_values = model_sm.pvalues

# Print all the results
print(f"Residual Sum of Squares (RSS): {RSS}")
print(f"Residual Standard Error (RSE): {RSE}")
print(f"Mean Squared Error (MSE): {MSE}")
print(f"R-squared (R2) statistic: {R2}")

print("\nStandard Errors for each feature:")
for feature, se in zip(X_sel.columns, standard_errors[1:]):  # Skipping the intercept
    print(f"{feature}: {se}")

print("\nT-statistics for each feature:")
for feature, t in zip(X_sel.columns, t_statistics[1:]):  # Skipping the intercept
    print(f"{feature}: {t}")

print("\nP-values for each feature:")
for feature, p in zip(X_sel.columns, p_values[1:]):  # Skipping the intercept
    print(f"{feature}: {p}")
```

```
Residual Sum of Squares (RSS): 77.17661510648327
Residual Standard Error (RSE): 0.3088649324302553
Mean Squared Error (MSE): 0.09481156646988116
R-squared (R2) statistic: 0.6528397310358398

Standard Errors for each feature:
T_OR1: 0.8694665336716999
T_OR_Max1: 0.8672245002334331
T_FHC_Max1: 0.04429270282049222
T_FH_Max1: 0.048964051526189434

T-statistics for each feature:
T_OR1: 0.1763447758368738
T_OR_Max1: 0.48111454588726127
T_FHC_Max1: -2.049524154302016
T_FH_Max1: 7.545180752626417

P-values for each feature:
T_OR1: 0.860067221055746
T_OR_Max1: 0.630565185099945
T_FHC_Max1: 0.04073285925720997
T_FH_Max1: 1.216299743510608e-13
```

Figure 7: Results of calculations

3.11 Discarding features based on p-value

The p-values obtained for T_OR1 and T_OR_Max1 are greater than 0.05 (5%). Therefore those features can be discarded

4 Performance Evaluation of Linear Regression

4.1 Residual Standard Error (RSE) for Models A and B

Residual Standard Error (RSE) is calculated using :

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

Where:

- **SSE :** Sum of Squared Errors.
- **N :** Number of data samples
- **p :** Number of parameters in the model(including the intercept)

Data:

- **Model A :** SSE = 9, N = 10000, p = 3 (intercept + *w1,w2*)
- **Model B :** SSE = 2, N = 10000, p = 5 (intercept + *w1,w2,w3,w4*)

Calculations:

- **For Model A**

$$\text{RSE}_A = \sqrt{\frac{9}{10000-3}} = \sqrt{\frac{9}{9997}} \approx \sqrt{0.00090027} \approx 0.03$$

- **For Model B**

$$\text{RSE}_B = \sqrt{\frac{2}{10000-5}} = \sqrt{\frac{2}{9995}} \approx \sqrt{0.0002001} \approx 0.01414$$

4.2 Based on RSE, which model performs better?

Model A has a lower RSE (0.03)
Model B has a lower RSE (0.01414)
Model B performs better

4.3 Compute R-squared $\mathbf{R}^2 for Models A and B$

The R^2 value is calculated using the formula:

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

For Model A:

$$R_A^2 = 1 - \frac{9}{90} = 1 - 0.1 = 0.9$$

For Model B:

$$R_B^2 = 1 - \frac{2}{10} = 1 - 0.2 = 0.8$$

Conclusion: Model A has a higher R^2 value (0.9) compared to Model B (0.8), so Model A performs better in terms of R^2.