Contents:

Overview

In data preprocessing, it's crucial to handle features that have large variances or are on different scales. Without proper scaling, such features can distort machine learning models. Features with larger numerical ranges can dominate models like distance-based algorithms (e.g., KNN, SVM) or gradient-based algorithms (e.g., gradient descent in neural networks). To address this, **Min-Max Scaling** (also known as normalization) is applied. This scaling technique transforms features by scaling them into a fixed range, usually [0, 1].

Purpose of Scaling

The main purpose of scaling is to ensure that no feature disproportionately influences the model due to its scale. For example, if one feature has a range in the thousands while others are in the tens, the model might overemphasize the former. Scaling ensures that all features contribute equally to the learning process by bringing them into a common range.

Here are two reasons why scaling is important:

- **Variance Reduction**: Scaling reduces the effect of large variances and ensures that no feature disproportionately influences the model.
- **Model Efficiency**: It helps models converge faster and achieve better performance, especially for algorithms sensitive to the magnitude of feature values.

Code Implementation

```
# normalising data
# for attributes that have very large values
# do normalisation to reduce variance

# print("\nData Normalisation to handle attributes with very large values")

# select the numerical columns for scaling and returns a numeric DataFrame(featuresDF_numeric)
featuresDF_numeric = features_dataframe.select_dtypes(include='number')

# apply min-max scaling to the numerical columns(featuresDF_numeric-DataFrame of numerical columns)
scaled_dataFrame = MinMaxScaler().fit_transform(featuresDF_numeric)

# convert the scaled data back to a DataFrame with the original numerical column names
scaled_numerical_df = pd.DataFrame(scaled_dataFrame, columns=featuresDF_numeric.columns, index=features_d

# replace the original numerical columns in the features DataFrame with the scaled ones
features_dataframe[featuresDF_numeric.columns] = scaled_numerical_df

# add the 'Churn' column (target) back to the scaled dataset
scaled_df_final = pd.concat([features_dataframe, targetColumn], axis=1)

# display final scaled dataframe
scaled_df_final
```

Steps taken

Selecting Numerical Features: The code begins by selecting all numerical columns from features_dataframe using the select_dtypes(include='number') method. This ensures only numerical data is scaled, as non-numeric data like categorical variables do not require scaling.

Applying Min-Max Scaling: Next, we instantiate the MinMaxScaler() from sklearn.preprocessing and apply it to the numerical data. The fit_transform method adjusts the data to fit within the [0, 1] range.

Reconstructing the DataFrame: After scaling, the transformed data is returned as a NumPy array. This array is converted back into a Pandas DataFrame, preserving the original feature names and indices for seamless integration.

Replacing Original Data: The scaled data is then inserted back into the original features_dataframe, replacing the unscaled numerical values.

Reattaching the Target Column: Finally, the targetColumn (representing the 'Churn' or target feature) is concatenated with the newly scaled features, forming the complete processed dataset (scaled_df_final).

Results

Applying **Min-Max Scaling** transformed all numerical features into the range [0, 1]. This reduces the dominance of features with large magnitudes, making the dataset well-suited for machine learning models that are sensitive to feature scaling.

Before Scaling

	gender	SeniorCitizen	Dependents	tenure	PhoneService	MultipleLines	InternetService	MonthlyCharges	Contract_Month-to-month	Cont
0	0	0	0	1	0	0	0	29.85	1	
1	1	0	0	34	1	0	0	56.95	0	
2	1	0	0	2	1	0	0	53.85	1	
3	1	0	0	45	0	0	0	42.30	0	
4	0	0	0	2	1	0	1	70.70	1	
...	...	...	...	...	...	...	...	...	...	
6935	1	0	1	24	1	1	0	84.80	0	
6936	0	0	1	72	1	1	1	103.20	0	
6937	0	0	1	11	0	0	0	29.60	1	
6938	1	1	0	4	1	1	1	74.40	1	
6939	1	0	0	66	1	0	1	105.65	0	

6940 rows × 11 columns

As seen in the image above, the values contained in the two columns, tenure and monthly charges, can be seen to vary highly. This was before applying the scaling method on the dataset.

After Scaling

	gender	SeniorCitizen	Dependents	tenure	PhoneService	MultipleLines	InternetService	MonthlyCharges	Contract_Month-to-month	Co
0	0.0	0.0	0.0	0.013889	0.0	0.0	0.0	0.115423	1.0	
1	1.0	0.0	0.0	0.472222	1.0	0.0	0.0	0.385075	0.0	
2	1.0	0.0	0.0	0.027778	1.0	0.0	0.0	0.354229	1.0	
3	1.0	0.0	0.0	0.625000	0.0	0.0	0.0	0.239303	0.0	
4	0.0	0.0	0.0	0.027778	1.0	0.0	1.0	0.521891	1.0	
...	...	...	...	...	...	...	...	...	...	
6935	1.0	0.0	1.0	0.333333	1.0	1.0	0.0	0.662189	0.0	
6936	0.0	0.0	1.0	1.000000	1.0	1.0	1.0	0.845274	0.0	
6937	0.0	0.0	1.0	0.152778	0.0	0.0	0.0	0.112935	1.0	
6938	1.0	1.0	0.0	0.055556	1.0	1.0	1.0	0.558706	1.0	
6939	1.0	0.0	0.0	0.916667	1.0	0.0	1.0	0.869652	0.0	

6940 rows × 12 columns

As seen in the image above, the values contained in the two columns, tenure and monthly charges, have been scaled using the Min-Max Scaler function from the scikit-learn library. After applying **Min-Max Scaling**, all numerical features were transformed into the range [0, 1], ensuring that each feature contributes equally to the model's learning process. This is particularly beneficial for models that are sensitive to the magnitude of feature values.

Conclusion

Min-Max Scaling from the scikit-learn library is a simple yet powerful technique for normalizing data. By scaling numerical features to a fixed range, we ensure that all features contribute equally to the learning process. This technique effectively reduces the impact of large variances in the data, helping models perform better and converge faster and enhancing model accuracy and convergence speed, particularly in algorithms where feature magnitude impacts model performance.