LAB 01 REPORT

Feature Engineering

190304R
M.Nuwan Jayasanka
nuwanm.19@cse.mrt.ac.lk

Contents

Introduction

High-dimensional datasets can lead to problems like overfitting, increased computational costs, and decreased interpretability. Therefore, selecting the most relevant features or transforming the existing ones is essential to address these challenges and build efficient machine learning models. Feature engineering is the process of transforming raw data into meaningful and interpretable features to improve the performance of machine learning models. This report outlines the methodology and techniques employed to reduce and select features from the given dataset.

What is the Feature Engineering

Feature engineering is a fundamental practice in data preprocessing that involves manipulating, selecting, or creating new features from raw data to improve the performance of machine learning models. The process aims to enhance the representation of data, making it more suitable for modeling while preserving or amplifying the underlying patterns that are relevant to the task at hand.

in essence, feature engineering bridges the gap between raw data and effective model performance. It involves a combination of domain knowledge, creativity, and analytical skills to identify and transform the most informative aspects of the data. This can include tasks like scaling, normalization, encoding categorical variables, handling missing values, creating interaction terms, and extracting meaningful patterns from existing features.

The goal of feature engineering is to enable machine learning algorithms to learn from data more effectively, leading to improved predictive accuracy, faster convergence during training, and enhanced interpretability of model outcomes. While it requires a thoughtful and iterative approach, the impact of well-executed feature engineering can be profound, allowing models to unlock deeper insights and solve complex problems across various domains.

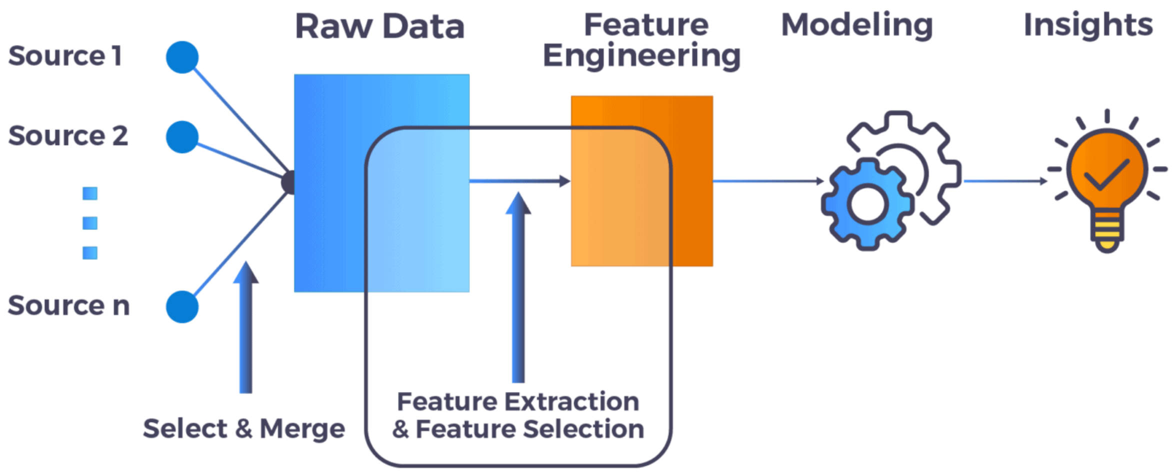


Figure 1Learning Process

Feature Engineering Process

Feature engineering involves a series of steps aimed at refining the raw data to create informative and effective features for machine learning models. Here's a concise overview of the process:

1. Understanding the Problem and Data
2. Exploratory Data Analysis (EDA):
3. Feature Selection
4. Feature Transformation
5. Feature Creation
6. Domain-Specific Feature Engineering
7. Regularization and Feature Importance
8. Iterative Process
9. Model Evaluation
10. Interpretability and Validation
11. Fine-Tuning

Used Methods

Feature Engineering

SelectKBest Method

The SelectKBest method is a feature selection technique used in feature engineering to identify the most important features in a dataset. It assigns scores to features based on a chosen scoring function and selects the top 'k' features with the highest scores. By retaining only the most relevant features, SelectKBest helps improve model performance, reduces overfitting, and enhances model interpretability. The choice of scoring function and the value of 'k' depend on the data and the problem at hand. This method is commonly employed in machine learning tasks to streamline the feature space and optimize model predictions.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique used in feature engineering to transform high-dimensional data into a lower-dimensional space while preserving as much variance as possible. It involves finding principal components, which are linear combinations of original features, and selecting the most informative ones. PCA is valuable for reducing overfitting, visualizing data, and emphasizing signal over noise. However, it can lead to loss of feature interpretability. It's implemented through steps like mean-centering, eigenvalue decomposition, and projection. Libraries like scikit-learn provide easy-to-use PCA implementations for enhancing data analysis and modeling efficiency.

Model Training

Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful supervised learning algorithms used for classification and regression tasks. They seek to find a hyperplane that best separates data points of different classes while maximizing the margin between them. SVM can handle complex decision boundaries using kernel functions, allowing for effective handling of nonlinear data. Its emphasis on maximizing the margin often leads to better generalization and performance. SVM is implemented in Python libraries like scikit-learn and is particularly valuable in high-dimensional spaces. However, proper parameter tuning and feature scaling are crucial for optimal results.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a versatile supervised machine learning algorithm used for classification and regression tasks. It predicts labels or values based on the similarity between data points and their 'k' nearest neighbors in the feature space. KNN's simplicity and adaptability make it suitable for various applications. It's particularly effective for capturing nonlinear relationships and complex decision boundaries. However, careful selection of the 'k' parameter and appropriate preprocessing, like feature scaling, are essential for achieving accurate results.

Other Methods

Standard Scaler

The StandardScaler is an essential preprocessing method frequently employed in machine learning to normalize and standardize features in a dataset. By adjusting the features to exhibit a mean of zero and a standard deviation of one, the scaler ensures that the data is on a uniform scale, which is particularly advantageous for algorithms that are sensitive to the varying magnitudes of features. This process aids in preventing certain features from disproportionately affecting model training and performance, facilitating the convergence of optimization algorithms and enhancing the efficacy of distance-based techniques. The StandardScaler's role in the data preprocessing pipeline is pivotal, contributing to more reliable and effective model outcomes by aligning feature distributions and mitigating the risk of biased learning due to disparate feature scales.

The provided code snippet performs data preprocessing using the StandardScaler from scikit-learn on multiple datasets for a multi-label classification task. Labels are defined as constants, and the feature columns are generated based on their indices. The code reads training, validation, and test data from CSV files. For each target label (L1, L2, L3, L4), the code applies feature scaling separately. If the target label is 'label_2', a specific subset of the dataset is considered. The scaler is used to standardize the features within each dataset. The standardized features are then stored in separate dictionaries for training, validation, and testing sets corresponding to each target label. This code segment is pivotal in preparing the data for subsequent machine learning model training and evaluation.

Evaluation Metrics

Accuracy

This metric gauge the overall correctness of a model's predictions by measuring the proportion of correctly classified instances out of the total. While intuitive, Accuracy can be misleading when dealing with imbalanced datasets, as it doesn't account for false positives and false negatives.

Precision

Precision emphasizes the quality of positive predictions. It quantifies the ratio of true positives (correctly predicted positives) to the total instances predicted as positive. High Precision signifies low false positive rate, making it invaluable when misclassification costs are high.

Recall

Recall, also known as Sensitivity or True Positive Rate, focuses on the model's ability to capture all actual positive instances. It calculates the ratio of true positives to the total actual positive instances. High Recall indicates a low false negative rate, which is crucial when failing to predict positives can be costly.

Results

Label_1

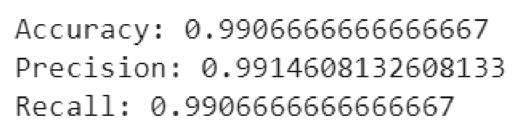
Model train method

Support Vector Machine (SVM)

Feature selection method

- SelectKBest

Accuracy Before Feature Engineering



```
Accuracy: 0.9906666666666667
Precision: 0.9914608132608133
Recall: 0.9906666666666667
```

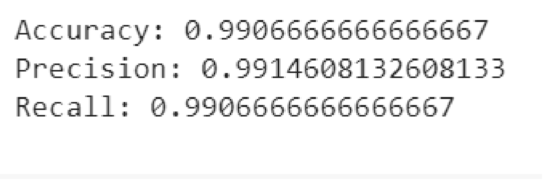
Accuracy After Feature Engineering

State	Num_of Features	Evaluation Metrics
SelectKBest(f_classif, k=100)	100 Features	Accuracy: 0.976 Precision: 0.9774185666185666 Recall: 0.976

SelectKBest(f_classif, k=75)	75 Features	Accuracy: 0.972 Precision: 0.9736650570275958 Recall: 0.972

- Principal Component Analysis (PCA)

Accuracy Before Feature Engineering



```
Accuracy: 0.9906666666666667
Precision: 0.9914608132608133
Recall: 0.9906666666666667
```

Accuracy After Feature Engineering

State	Num_of Features	Evaluation Metrics
`PCA(n_components = 0.95, svd_solver ='full')`	67 features	Accuracy: 0.9813333333333333 Precision: 0.9835733337182255 Recall: 0.9813333333333333
`PCA(n_components = 0.85, svd_solver ='full')`	39 features	Accuracy: 0.932 Precision: 0.9384671976687456 Recall: 0.932

Label_2

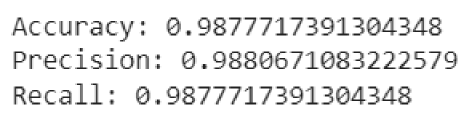
Model train method

K-Nearest Neighbors Algorithm (KNN)

Feature selection method

- Principal Component Analysis (PCA)

Accuracy before Feature Engineering



```
Accuracy: 0.9877717391304348
Precision: 0.9880671083222579
Recall: 0.9877717391304348
```

Accuracy after Feature Engineering

State	Num_of Features	Evaluation Metrics
`PCA(n_components = 0.75, svd_solver ='full')`	25 features	Accuracy: 0.9605978260869565 Precision: 0.9620553181192202 Recall: 0.9605978260869565
`PCA(n_components = 0.85, svd_solver ='full')`	39 features	Accuracy: 0.9823369565217391 Precision: 0.9829790686548117 Recall: 0.9823369565217391
`PCA(n_components = 0.95, svd_solver ='full')`	67 features	Accuracy: 0.9877717391304348 Precision: 0.9880990200464284 Recall: 0.9877717391304348

Label_3

Model train method

Support Vector Machine (SVM)

Feature selection method

- SelectKBest

Accuracy before Feature Engineering

```
Accuracy: 0.9986666666666667
Precision: 0.9986759906759908
Recall: 0.9986666666666667
```

Accuracy after Feature Engineering

State	Num_of Features	Evaluation Metrics
`SelectKBest(f_classif, k=5)`	5 features	Accuracy: 0.98 Precision: 0.9803982355794724 Recall: 0.9986666666666667
`SelectKBest(f_classif, k=15)`	15 features	Accuracy: 0.984 Precision: 0.9841076274294095 Recall: 0.9986666666666667

- Principal Component Analysis (PCA)

Accuracy before Feature Engineering

```
Accuracy: 0.9986666666666667
Precision: 0.9986759906759908
Recall: 0.9986666666666667
```

Accuracy after Feature Engineering

State	Num_of Features	Evaluation Metrics
`PCA(n_components = 0.95, svd_solver ='full')`	67 features	Accuracy: 0.9986666666666667 Precision: 0.9986688560481665 Recall: 0.9986666666666667
`PCA(n_components = 0.85, svd_solver ='full')`	39 features	Accuracy: 0.9973333333333333 Precision: 0.9973420765027322 Recall: 0.9973333333333333
`PCA(n_components = 0.75, svd_solver ='full')`	25 features	Accuracy: 0.996 Precision: 0.9960164514233708 Recall: 0.996
`PCA(n_components = 0.65, svd_solver ='full')`	17 features	Accuracy: 0.9933333333333333 Precision: 0.9933569121707508 Recall: 0.9933333333333333
`PCA(n_components = 0.45, svd_solver ='full')`	9 features	Accuracy: 0.9866666666666667 Precision: 0.986616705698673 Recall: 0.9866666666666667
`PCA(n_components = 0.35, svd_solver ='full'`	6 features	Accuracy: 0.9533333333333334 Precision: 0.9528209158420753 Recall: 0.9533333333333334

Label_4

Model train method

K-Nearest Neighbors Algorithm (KNN)

Feature selection method

- Principal Component Analysis (PCA)

Accuracy before Feature Engineering

```
Accuracy: 0.9946666666666667
Precision: 0.9947064676616915
Recall: 0.9946666666666667
```

Accuracy after Feature Engineering

State	Num_of Features	Evaluation Metrics

`PCA(n_components = 0.75, svd_solver ='full')`	25 features	`Accuracy: 0.976`
		`Precision: 0.9766319451841071`
		`Recall: 0.976`

`PCA(n_components = 0.85, svd_solver ='full')`	39 features	`Accuracy: 0.988`
		`Precision: 0.9882956567537573`
		`Recall: 0.988`

`PCA(n_components = 0.95, svd_solver ='full')`	67 features	`Accuracy: 0.992`
		`Precision: 0.992089219330855`
		`Recall: 0.992`

References

Note Book - Source Code