CS 4622 – Machine Learning
Lab 01- Feature Engineering

190199A – S Gopinath

Colab Link

Dropping constant features

Initially I thought to remove the features which have constant features which are not important for solving the problem statement.

Using Variance Threshold: Features with very low variance provide little information and may not contribute much to the model's performance. The idea is to set a threshold for the variance and remove features that have variance below that threshold.

However, the provided dataset didn't include low variance features. So, I couldn't use this method in this problem.

Dropping features with the help of Correlation

Identifying and handling highly correlated features is an important step in preprocessing data for machine learning. Highly correlated features can introduce multicollinearity, which might negatively affect model performance and interpretability.

I processed the dataset and extracted correlation matrix. I visualized the correlation matrix using heatmap.

I couldn't identify strong correlations between variables. So I couldn't use this method also.

Information gain

I used mutual information In Regression Problem Statements which is a Filter method used in feature selection.

Mutual Information: Estimate mutual information for a continuous target variable.

Mutual information (MI) between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

The function relies on nonparametric methods based on entropy estimation from k-nearest neighbors distances. Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.

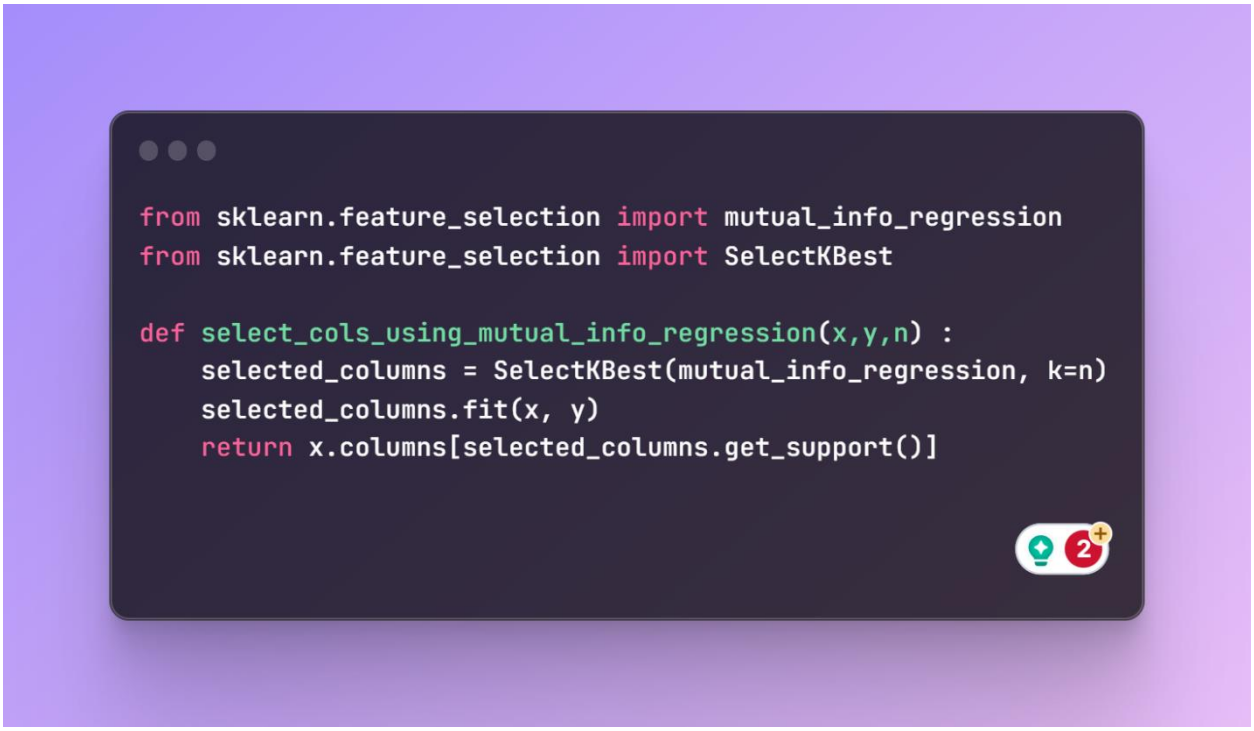
Inshort A quantity called mutual information measures the amount of information one can obtain from one random variable given another.

The mutual information between two random variables X and Y can be stated formally as follows:

$$I(X\,;\,Y) \;=\; H(X) - H(X \mid Y)$$

Where I(X ; Y) is the mutual information for X and Y, H(X) is the entropy for X and H(X | Y) is the conditional entropy for X given Y. The result has the units of bits.

I successfully used this method to pick the best performing features using SelectKBest function. By using this method, it was possible to extract the features without big loss in accuracy.



```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectKBest

def select_cols_using_mutual_info_regression(x,y,n) :
    selected_columns = SelectKBest(mutual_info_regression, k=n)
    selected_columns.fit(x, y)
    return x.columns[selected_columns.get_support()]
```

Here ‘n’ specifies how many features needed.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving as much of the original data's variance as possible. It's commonly used for feature extraction, data visualization, and noise reduction. PCA identifies the principal components (linear combinations of the original features) that capture the most significant variations in the data.

Scaling features before PCA is important to ensure that all features are treated equally, prevent features with larger scales from dominating the analysis, enable meaningful interpretation of principal components, improve numerical stability, and achieve effective dimensionality reduction.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

def perform_PCA(x,x_valid,n):
    scaler = StandardScaler()
    scaler.fit_transform(x)
    scaled_x = scaler.transform(x)
    scaled_x_valid = scaler.transform(x_valid)
    pca = PCA(n_components=n)
    pca.fit(scaled_x)
    x_pca = pca.transform(scaled_x)
    x_pca_valid = pca.transform(scaled_x_valid)
    pca_df = pd.DataFrame(data=x_pca)
    pca_valid_df = pd.DataFrame(data=x_pca_valid)
    return pca_df, pca_valid_df
```

Here we receive both X_Train & X_Test or X_Valid data frames and perform PCA Analysis and return resulted data frames.

Merging the two methods

- I thought to first remove unnecessary features using the Information Gain method.
- Using the Information gain method, I reduced the features to 100 from 256.
- Then I used PCA to transform and reduce the dimension of the features to lower dimension.

I couldn't use the Wrapper method & Intrinsic method due to lower performance of my laptop.

Initially I went with SVM algorithm. However due to bad accuracy results for some labels, I need to switch to another algorithm. Then I decided to go with k nearest neighbor algorithm. It gave impressive results where accuracy levels were close to 100% with all 255 features.

Obtained Accuracy results with valid data given below.

Label	Metrices	Results with 255 features	Number of features after Feature Engineering	Results after Feature Engineering
Label 1	Accuracy	0.987	40	0.976
	Precision	0.988		0.979
	Recall	0.987		0.976
Label 2	Accuracy	0.988	40	0.973
	Precision	0.988		0.974
	Recall	0.988		0.973
Label 3	Accuracy	1	15	1
	Precision	1		1
	Recall	1		1
Label 4	Accuracy	0.993	30	0.983
	Precision	0.993		0.983
	Recall	0.993		0.983

It is possible to archive good feature reduction without losing valuable data which is clearly shown in the above table.