**Homework – 5**

Name – Tapan Khaladkar

NetID – tk3301

1. **Data Preprocessing**

Data Preprocessing and Feature Importance Analysis Report

Introduction

I have outlined a comprehensive approach to data preprocessing and feature importance analysis using machine learning techniques in Python. This report summarizes the key steps and findings from the script, which include handling missing values, identifying and visualizing outliers, scaling and normalization, principal component analysis (PCA), label encoding, and feature importance extraction using a Random Forest Classifier.

Data Preprocessing

Missing Values

I began with the identification of missing values in the dataset:

```python

missing\_values = data.isnull().sum()

```

This step is crucial as missing data can significantly impact the quality of the model. However, specific actions taken to handle these missing values (such as imputation or deletion) are not detailed in the code provided.

Outlier Detection

Outliers can adversely affect model performance, especially in regression-based algorithms. The script uses the Interquartile Range (IQR) method to detect outliers in each numerical column:

```python

for column in numerical\_columns:

q1 = data[column].quantile(0.25)

q3 = data[column].quantile(0.75)

iqr = q3 - q1

...

outliers\_count = data.loc[(data[column] < fence\_low) | (data[column] > fence\_high)].shape[0]

```

Each outlier count is recorded, providing a clear picture of potential anomalies in the data that may require further attention.

Feature Scaling and Normalization

I have used both StandardScaler and MinMaxScaler to preprocess the data, targeting specific features based on their distribution:

```python

transformed\_data[columns\_to\_standardize] = scaler\_standard.fit\_transform(data[columns\_to\_standardize])

transformed\_data[columns\_to\_normalize] = scaler\_minmax.fit\_transform(data[columns\_to\_normalize])

```

Standardization is applied to features that are normally distributed, while normalization is used for skewed distributions, optimizing them for better performance in machine learning models.

Principal Component Analysis (PCA)

PCA is applied to reduce the dimensionality of the dataset, which helps in visualizing complex datasets and can improve model efficiency:

```python

pca = PCA(n\_components=2)

principal\_components = pca.fit\_transform(numerical\_data)

```

This step results in a new dataset with principal components that capture the most significant variance in the data with fewer dimensions.

Feature Importance from Random Forest

Model Building and Evaluation

A Random Forest Classifier is trained on the preprocessed data, and its performance is evaluated:

```python

rf\_classifier.fit(X\_train, y\_train)

accuracy = accuracy\_score(y\_test, y\_pred)

```

The model's accuracy is highlighted, providing an indication of how well the model performs on unseen data.

Feature Importance Analysis

This concludes with an analysis of feature importance derived from the Random Forest model:

```python

feature\_importance = rf\_classifier.feature\_importances\_

```

A visual representation is created to show the importance of each feature, helping to identify which features have the most impact on the target variable. This insight is valuable for understanding the driving factors in the dataset and can guide further model refinement and feature engineering.

Conclusion

The preprocessing steps outlined in the script are essential for preparing the data for modeling. They help in improving model accuracy and robustness by addressing potential issues like missing values, outliers, and feature scaling. The feature importance analysis provides critical insights into the most influential features, which can be crucial for business decisions and further data analysis. Future recommendations include detailed handling of missing data, further exploration of model parameters, and continuous monitoring and updating of the preprocessing steps based on model performance and changing data characteristics.

1. **Modeling**
2. **KNN** –

Precision for class 0 (Alive) is 89%, and for class 1 (Dead) is 75%.

Recall for class 0 is 98%, but for class 1, it is quite low at 34%. This indicates the model is much better at predicting the majority class (Alive) than the minority class (Dead).

F1-score for class 0 is 94% and for class 1 is 47%.

The model is performing well in terms of overall accuracy and is excellent at identifying the Alive status.

However, the ability to correctly identify the Dead status (recall for class 1) is relatively poor. This might be due to the imbalance in the dataset where the Alive cases dominate.

1. **Naïve Bayes –**

Precision for class 0 (Alive) is 91%, and for class 1 (Dead) is 46%.

Recall for class 0 is 90%, and for class 1, it's 49%.

F1-score for class 0 is 91% and for class 1 is 48%.

The overall accuracy of the Naive Bayes model is slightly lower than the KNN model.

The precision and recall for the minority class (Dead) have improved compared to the KNN model, indicating a better balance in identifying both classes, although the performance for predicting the Dead class remains modest.

1. **Decision Tree –**

Precision for class 0 (Alive) is 92%, and for class 1 (Dead) is 53%.

Recall for class 0 is 91%, and for class 1, it's 56%.

F1-score for class 0 is 92% and for class 1 is 54%.

The Decision Tree model shows a good balance between precision and recall, with better performance on predicting the minority class (Dead) compared to the previous Naive Bayes model.

The overall accuracy is also competitive, suggesting that the model does a good job at distinguishing between the classes.

1. **Random Forest –**

Precision for class 0 (Alive) is 92%, and for class 1 (Dead) is 84%.

Recall for class 0 is 98%, and for class 1, it's 52%.

F1-score for class 0 is 95% and for class 1 is 64%.

The Random Forest model shows excellent overall accuracy and high precision for both classes.

The recall for class 1 (Dead) has improved compared to single decision trees, although it remains modest, indicating that the model still struggles somewhat with detecting the minority class effectively.

1. **Gradient Boosting –**

Precision for class 0 (Alive) is 92%, and for class 1 (Dead) is 85%.

Recall for class 0 is 98%, and for class 1, it's 53%.

F1-score for class 0 is 95% and for class 1 is 65%.

Gradient Boosting shows an excellent overall accuracy and a good precision for both classes.

The model has a high recall for the majority class (Alive), but the recall for the minority class (Dead) still shows room for improvement, similar to the Random Forest model.

**Results –**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Accuracy | Confusion  Matrix | Precision | Recall | F1 – Score | Macro Avg | Weighted Avg |
| Kmeans | 88.44% | [[671, 14], [79, 41]] | [0.89, 0.75] | [0.98, 0.34] | [0.94, 0.47] | [0.82, 0.66, 0.70] | [0.87, 0.88, 0.87] |
| Naïve Bayes | 83.98% | [[617, 68], [61, 59]] | [0.91, 0.46] | [0.90,  0.49] | [0.91, 0.48] | [0.69, 0.70, 0.69] | [0.84, 0.84, 0.84] |
| Decision  Tree | 86.09% | [[626, 59], [53, 67]] | [0.92, 0.53] | [0.91, 0.56] | [0.92, 0.54] | [0.73, 0.74, 0.73] | [0.86, 0.86, 0.86] |
| Random Forest | 91.30% | [[673, 12], [58, 62]] | [0.92, 0.84] | [0.98, 0.52] | [0.95, 0.64] | [0.88, 0.75, 0.79] | [0.91, 0.91, 0.90] |
| Gradient Boosting | 91.55% | [[674, 11], [57, 63]] | [0.92, 0.85] | [0.98, 0.53] | [0.95, 0.65] | [0.89, 0.75, 0.80] | [0.91, 0.92, 0.90] |

**Hyperparameter Search –**

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Out[15]:

({'n\_estimators': 50,

'min\_samples\_split': 10,

'min\_samples\_leaf': 1,

'max\_depth': None,

'bootstrap': True},

0.9018300281096954,

{'n\_estimators': 100,

'min\_samples\_split': 5,

'min\_samples\_leaf': 2,

'max\_depth': 3,

'learning\_rate': 0.1},

0.8999676400413434)