EC – 9560 DATA MINING

LAB 04

ATPUTHARAVI.R

2020/E/015

18 DECEMBER 2024

SUMMARY OF UNTIL LAST LAB SESSION:

1. Data Download and Merge:
   - Downloaded datasets (`train_dataset.csv` and `test_dataset.csv`) using Kaggle API.
   - Included dataset column in identifying train from test data.
   - Merged train and test datasets in an all-in-one DataFrame for pre-processing.

2. First Exploration over the Data:
   - Display some general info about the dataset by issuing .info() and .head().
   - Check for missing values and duplicates.

3. Data Cleaning:
   - Deleted duplicate rows from the dataset.
   - Checked for and managed all columns with null entries to deal with missing values.

4. Outlier Detection and Elimination:
   - Removed and detected outliers for all numerical columns using IQR.

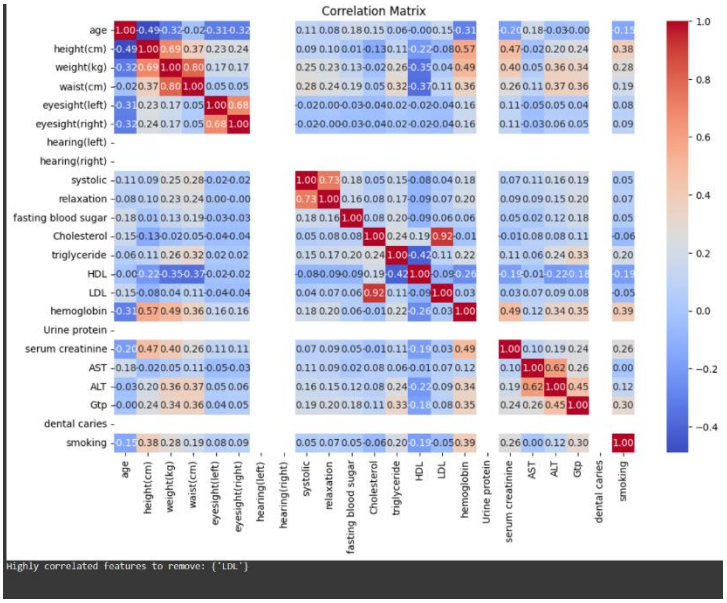
5. Standardization:
   - Standardized using StandardScaler in such a way that all numerical features are standardized to have a mean of 0 and a variance of 1. Important for using models like SVMs, and Logistic Regression since they are sensitive to the scaling of the features.

This exhaustive pre-processing ensured that we obtained a clear dataset normalized and ready for training and evaluating all models.

STEP 1: Feature selection

- Correlation-based filtering is a feature selection technique to decrease the dimensionality of the data set by identifying and eliminating features that are highly correlated with each other. The aim is to keep the most informative features while eliminating the redundant ones that might lower the performance of machine learning models.

```
# Exclude non-numerical columns
numeric_df = df_no_outliers.select_dtypes(include=[np.number])

# Compute the correlation matrix
correlation_matrix = numeric_df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()

# Set a threshold for high correlation (e.g., 0.8)
high_correlation_threshold = 0.8
high_corr_features = set()

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > high_correlation_threshold:
            high_corr_features.add(correlation_matrix.columns[i])

print("Highly correlated features to remove:", high_corr_features)

# Drop highly correlated features
df_no_outliers = df_no_outliers.drop(columns=high_corr_features)
```



STEP 2: Drop the dataset column

- It should ensure that they're purely numerical features and the target variable for modeling.

```
from sklearn.model_selection import train_test_split

# Drop the 'dataset' column
X = df_no_outliers.drop(columns=['smoking', 'dataset'])  # Replace 'smoking' with the actual target column name
y = df_no_outliers['smoking']
```

STEP 3: Convert categorical target variable to numerical

- If the target variable (y) contains string labels, use LabelEncoder to encode them into numerical forms.

```
from sklearn.preprocessing import LabelEncoder

# Encode the target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

STEP 4: Split data

- Proceed with splitting the dataset into training and testing sets.

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

STEP 5: Train multiple models and Cross-Validation for all models

- Train different models such as Logistic Regression, Random Forest, Gradient Boosting, SVM, K-NN, Naïve Bayes and Decision Tree.
- I introduced cross-validation of 5-fold across datasets with cross_val_score function. The scoring parameter calculates accuracy, precision, recall, and F1 scores for each model.
- make_scorer specifies custom scoring-for precision, recall, and F1-scoreaveraged across classes for multiclass classification.

```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
import pandas as pd

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'SVM': SVC(),
    'K-NN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier()
}

# Initialize an empty list to store results
results = []

# Define a cross-validation function
def evaluate_model(model, X_train, y_train):
    # Use cross-validation to get scores
    accuracy = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy').mean()
    precision = cross_val_score(model, X_train, y_train, cv=5, scoring=make_scorer(precision_score, average='macro')).mean()
    recall = cross_val_score(model, X_train, y_train, cv=5, scoring=make_scorer(recall_score, average='macro')).mean()
    f1 = cross_val_score(model, X_train, y_train, cv=5, scoring=make_scorer(f1_score, average='macro')).mean()

    return accuracy, precision, recall, f1
```

```python
# Evaluate each model using cross-validation
for name, model in models.items():
    accuracy, precision, recall, f1 = evaluate_model(model, X_train, y_train)

    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-score': f1
    })
```

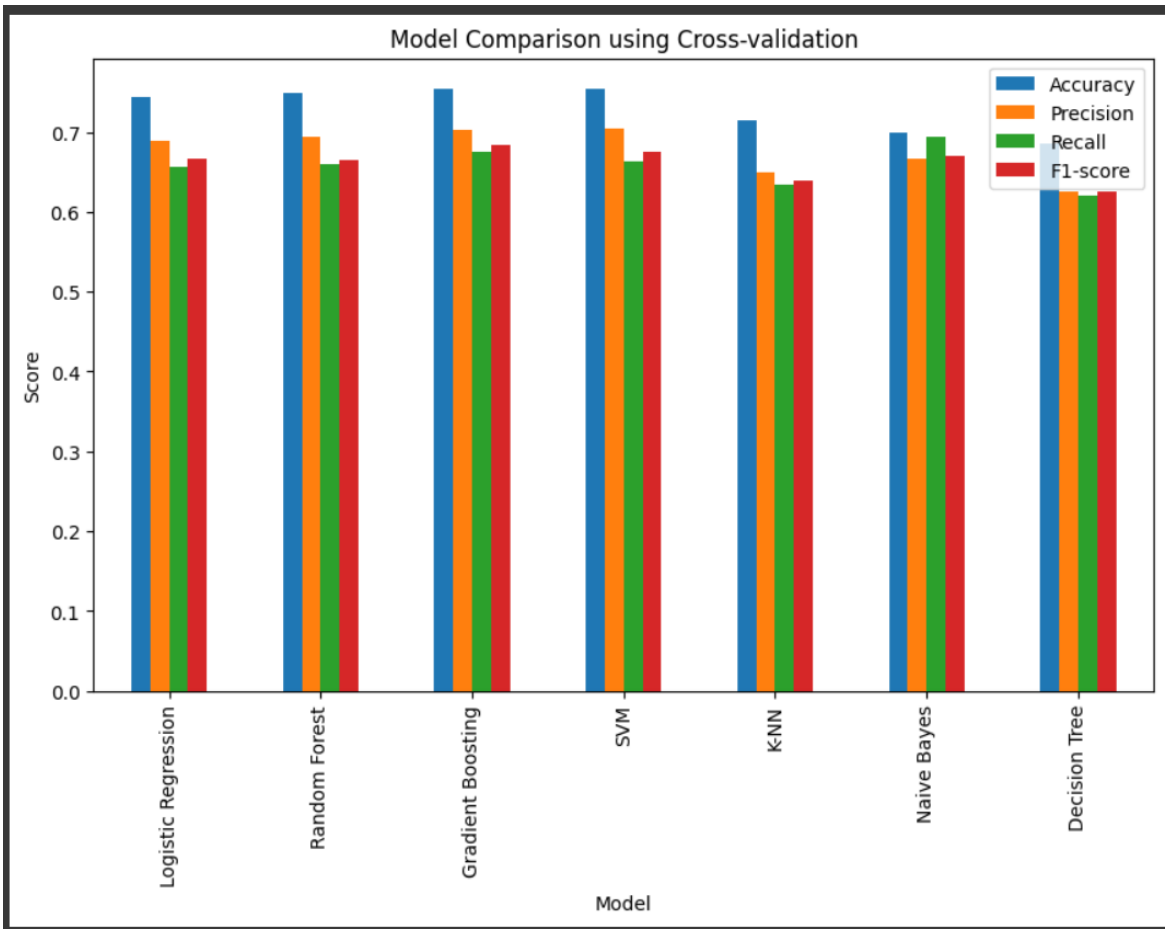
STEP 6: Model comparison analysis

```python
# Convert results to a DataFrame for easy comparison
results_df = pd.DataFrame(results)
print(results_df)
```

	Model	Accuracy	Precision	Recall	F1-score
0	Logistic Regression	0.744461	0.689346	0.655763	0.665805
1	Random Forest	0.748391	0.694099	0.659687	0.665341
2	Gradient Boosting	0.753909	0.702541	0.675012	0.684548
3	SVM	0.754577	0.705123	0.663434	0.675290
4	K-NN	0.714357	0.649070	0.633327	0.638852
5	Naive Bayes	0.698637	0.667249	0.694203	0.669947
6	Decision Tree	0.685676	0.625331	0.619804	0.625148

STEP 7: Visualize model performance

```
# Visualize the model comparison
results_df.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title('Model Comparison using Cross-validation')
plt.ylabel('Score')
plt.show()
```



The different models can be compared with respect to the accuracy, precision, recall and F1-score figures derived from this study. Here is a brief analysis of these parameters:

- Accuracy: This is the ratio of right predictions to total predictions made. It is often preferred to have accurate results; accuracy becomes misleading when data is not balanced.
- Precision: Precision refers to the ratio of true positives of all positive predictions made. Higher precision value means less false positive case.
- Recall: The proportion of true positive outcomes. Higher "recall" means lower "false negatives".
- F1 Score: It is of harmonic mean of precision and recall, thus providing balance between them. It means better balance between precision and recall with a higher F1 score.

CONCLUSION

Best Model:

Among the best models in terms of accuracy, precision, recall, and F1-score are Gradient Boosting and SVM.

- Gradient Boosting has the highest F1 score (0.684548) over all models and hence depicted a good balance between precision and recall.
- SVM shows performance comparable to Gradient Boosting; however, it is slightly lower in terms of F1 score (0.675290) and recall (0.663434).

Based on these findings, **Gradient Boosting** would be the most suitable model for this problem.

Future Improvements:

- Deep learning models like neural networks will improve accuracy as more complex relationships may be present in the data.
- Integration of other features of lifestyle, genetic data, or environmental factors might yield improvements in predictions.
- Hyperparameter tuning of models using grid search or random search might improve performance as well.

This project produced a very nice prediction machine learning pipeline that predicts smoking status with the Gradient Boosting model performing the best. Future work might explore data source possibilities, advanced model experimentation, or now optimizing models for even better performance.