Report of Q5
Smoke Status Recognition

50015627 JIANG Zhuoyang

1. **Question Description:**

Smoking is one of the major health problems. From a biomedical point of view, we can determine whether a patient smokes from certain biometric information. I was required to implement a binary algorithm to predict a patient's smoking status given information about various other health indicators.

- Load Data
- Preprocessing and EDA
- Feature Engineering
- Model Training
- Use Model to Do Classification On Test set

2. Load Data Especially Video Data

Load Training Data:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159256 entries, 0 to 159255
Data columns (total 24 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   id                   159256 non-null  int64
 1   age                  159256 non-null  int64
 2   height(cm)           149701 non-null  float64
 3   weight(kg)           159256 non-null  int64
 4   waist(cm)            159248 non-null  float64
 5   eyesight(left)       149692 non-null  float64
 6   eyesight(right)      159247 non-null  float64
 7   hearing(left)        159256 non-null  int64
 8   hearing(right)       149701 non-null  float64
 9   systolic             159256 non-null  int64
 10  relaxation           159256 non-null  int64
 11  fasting blood sugar  159256 non-null  int64
 12  Cholesterol          159256 non-null  int64
 13  triglyceride         159256 non-null  int64
 14  HDL                  159256 non-null  int64
 15  LDL                  159256 non-null  int64
 16  hemoglobin           159256 non-null  float64
 17  Urine protein        149669 non-null  float64
 18  serum creatinine     159256 non-null  float64
 19  AST                  159256 non-null  int64
 20  ALT                  159256 non-null  int64
 21  Gtp                  159256 non-null  int64
 22  dental caries        159256 non-null  int64
 23  smoking              159256 non-null  int64
dtypes: float64(8), int64(16)
memory usage: 29.2 MB
```

Figure.1 Training Data Information

Load Test Data:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106171 entries, 0 to 106170
Data columns (total 23 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   id                   106171 non-null  int64
 1   age                  106171 non-null  int64
 2   height(cm)           106171 non-null  int64
 3   weight(kg)           106171 non-null  int64
 4   waist(cm)            106171 non-null  float64
 5   eyesight(left)       106171 non-null  float64
 6   eyesight(right)      106171 non-null  float64
 7   hearing(left)        106171 non-null  int64
 8   hearing(right)       106171 non-null  int64
 9   systolic             106171 non-null  int64
 10  relaxation           106171 non-null  int64
 11  fasting blood sugar  106171 non-null  int64
 12  Cholesterol          106171 non-null  int64
 13  triglyceride         106171 non-null  int64
 14  HDL                  106171 non-null  int64
 15  LDL                  106171 non-null  int64
 16  hemoglobin           106171 non-null  float64
 17  Urine protein        106171 non-null  int64
 18  serum creatinine     106171 non-null  float64
 19  AST                  106171 non-null  int64
 20  ALT                  106171 non-null  int64
 21  Gtp                  106171 non-null  int64
 22  dental caries        106171 non-null  int64
dtypes: float64(5), int64(18)
memory usage: 18.6 MB
```

Figure.2 Test Data Information

3. Preprocessing and EDA

(1) Missing Processing:

Fill missing values using the mean

```
# Fill missing values using the mean
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
X_test_imputed = imputer.fit_transform(X_test)
```

Figure.3 Fill missing values using the mean

(2) EDA

Draw box plots and aCorrelation Heatmap for all features, we can find that there are features in different distributions, and some of them have relationship we can do different kind of Feature Engineering on features with different distributions and use the relationship between features to generate new features.

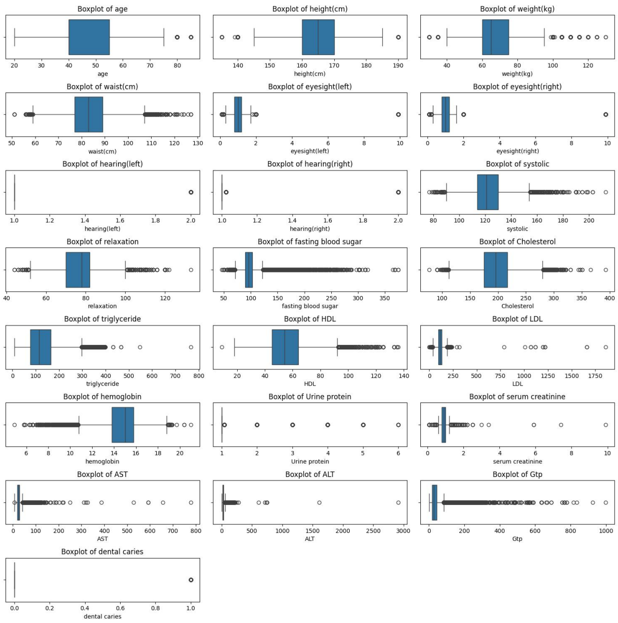

Figure.4 Draw a box plot of all features

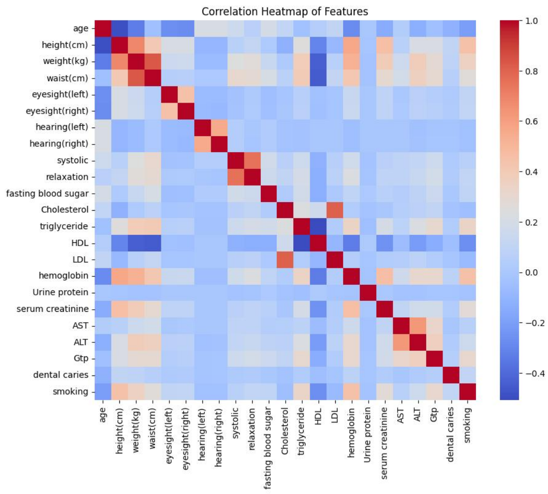

Figure.5 Correlation Heatmap of Features

4. Feature Engineering

Create with feature as below:

➢ Create new feature - BMI:

$$\text{BMI} = \text{weight\_kg} / ((\text{height\_cm} / 100)^2)$$

➢ Create new feature - Average Eyesight

$$\text{average\_eyesight} = (\text{left\_eyesight} + \text{right\_eyesight}) / 2$$

➢ Create new feature - Average Hearing

$$\text{average\_hearing} = (\text{left\_hearing} + \text{right\_hearing}) / 2$$

Do Feature Engineering for both Training and Test data:

```python
# Concatenate training and test data without labels
all_data = pd.concat([train_data.iloc[:, 1:-1], test_data.iloc[:, 1:-1]], axis=0)

# Impute missing values using mean strategy
imputer = SimpleImputer(strategy='mean')
all_data_imputed = imputer.fit_transform(all_data)
```

```python
# Create new feature: BMI
height_cm = all_data_imputed[:, all_data.columns.get_loc('height(cm)')]
weight_kg = all_data_imputed[:, all_data.columns.get_loc('weight(kg)')]
BMI = weight_kg / ((height_cm / 100) ** 2)  # BMI calculation formula
all_data_imputed = np.column_stack((all_data_imputed, BMI.reshape(-1, 1)))

# Create new feature: Average Eyesight
left_eyesight = all_data_imputed[:, all_data.columns.get_loc('eyesight(left)')]
right_eyesight = all_data_imputed[:, all_data.columns.get_loc('eyesight(right)')]
average_eyesight = (left_eyesight + right_eyesight) / 2  # Average eyesight calculation formula
all_data_imputed = np.column_stack((all_data_imputed, average_eyesight.reshape(-1, 1)))

# Create new feature: Average Hearing
left_hearing = all_data_imputed[:, all_data.columns.get_loc('hearing(left)')]
right_hearing = all_data_imputed[:, all_data.columns.get_loc('hearing(right)')]
average_hearing = (left_hearing + right_hearing) / 2  # Average hearing calculation formula
all_data_imputed = np.column_stack((all_data_imputed, average_hearing.reshape(-1, 1)))
```

Figure.7 Input Organization of 2D Convolution Deep Model

The Number of features after feature engineering shows as below:

```
Number of features in training set: 25
Number of features in testing set: 25
```

Figure.7 Number of features after feature engineering

5. Model Training and Evaluation

(1) Method Description - LightGBM :

The LGBMClassifier is a classification algorithm provided by the LightGBM library, which stands for Light Gradient Boosting Machine. It's based on the gradient boosting framework and is specifically designed for handling large datasets efficiently while delivering high performance.

Specifically, LightGBM's Gradient Boosting Framework means it constructed an ensemble of decision trees sequentially to minimize the errors made by the preceding trees.

It's optimized for speed and memory efficiency using a histogram-based approach for binning continuous feature values and making split decisions vertically on these bins, reducing memory usage and speeding up training.

Unlike traditional depth-wise tree growth, LightGBM grows trees leaf-wise. It selects the leaf with the maximum delta loss to grow, allowing for a more complex tree and better capture of feature interactions.

What's more LightGBM can handle categorical features directly, avoiding the need for one-hot encoding and reducing memory overhead.

(2) Model Construction and GridSeach Defining:

Define the LGBMClassifier model and apply a parameters grid for automatically tuning, the GridSearch Method will help us find a set of good hyperparameter.

```python
# Define the parameters grid for tuning
param_grid = {
    'num_leaves': [20, 30, 40],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [5, 10, -1],   # -1 means no limit
    'min_child_samples': [20, 30, 50],
}

# Split the dataset into training and validation sets
X_train_improved, X_val_improved, y_train, y_val = train_test_split(X_improved, y, test_size=0.01, random_state=42)

# Initialize LightGBM classifier
lgb_classifier = lgb.LGBMClassifier(boosting_type='gbdt', objective='binary', metric='auc', n_jobs=-1, verbosity=1)

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=lgb_classifier, param_grid=param_grid, scoring='roc_auc', cv=5)
```

Figure.8 Model Construction and GridSearch

(3) Training Processing:

The training processing of LGBMClassifier can be should with the information:

```
In [25]:
# Fit GridSearchCV to find best parameters
grid_search.fit(X_train_improved, y_train)

# Get the best model from GridSearchCV
best_params = grid_search.best_params_
best_score = grid_search.best_score_
best_model = grid_search.best_estimator_
```

```
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2357
[LightGBM] [Info] Number of data points in the train set: 126131, number of used features: 25
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.437117 -> initscore=-0.252871
[LightGBM] [Info] Start training from score -0.252871
[LightGBM] [Info] Number of positive: 55134, number of negative: 70997
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.008740 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2364
[LightGBM] [Info] Number of data points in the train set: 126131, number of used features: 25
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.437117 -> initscore=-0.252871
[LightGBM] [Info] Start training from score -0.252871
[LightGBM] [Info] Number of positive: 68917, number of negative: 88746
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.011090 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2397
[LightGBM] [Info] Number of data points in the train set: 157663, number of used features: 25
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.437116 -> initscore=-0.252875
[LightGBM] [Info] Start training from score -0.252875
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Figure.9 Model Construction and GridSearch

(4) Tuning Result and Evaluation:

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 10, 'min_child_samples': 50, 'num_leaves': 30}
Best ROC-AUC Score: 0.865002584046359
Validation ROC-AUC Score: 0.8569691415960631
```

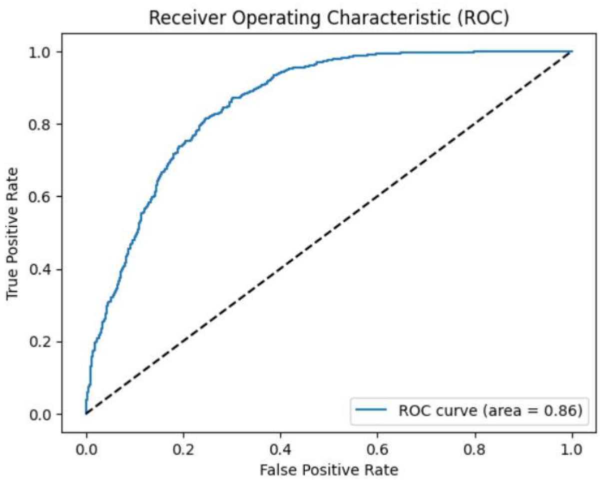

Figure.12 Tuning Result and Evaluation

6. Use Model to Do Classification On Test set

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106171 entries, 0 to 106170
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   id       106171 non-null  int64
 1   smoking  106171 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 1.6 MB
```

Figure.13 Output.csv