FinalModel

December 16, 2024

1 Final Model

Here the final best model based on all the past analysis and expiraments are implemented

[13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

plt.style.use('seaborn')
pd.set_option('display.max_columns', None)
```

```
/tmp/ipykernel_22002/762822019.py:11: MatplotlibDeprecationWarning: The seaborn
styles shipped by Matplotlib are deprecated since 3.6, as they no longer
correspond to the styles shipped by seaborn. However, they will remain available
as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn')
```

1.1 Data imported and feature engineered

[14]:
```
filename = 'Ndata.csv'
data = pd.read_csv(filename)
data.rename(columns={data.columns[0]: "id"}, inplace=True)
pd.set_option('display.max_columns', None) # To show all columns
data.sample(3)
```

[14]:

	id	ALT	HDL	Cholesterol	weight(kg)	LDL	age	hearing(left)	\
148122	148122	32	44	165	80	78	40	1	
115727	115727	30	67	191	75	86	20	1	
88007	88007	10	68	203	65	123	65	2	

	waist(cm)	hemoglobin	height(cm)	smoking
148122	88.0	15.8	180	1

```
115727      80.3        17.6         170        0
88007       84.0        13.7         150        0
```

[15]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159256 entries, 0 to 159255
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id              159256 non-null  int64
 1   ALT             159256 non-null  int64
 2   HDL             159256 non-null  int64
 3   Cholesterol     159256 non-null  int64
 4   weight(kg)      159256 non-null  int64
 5   LDL             159256 non-null  int64
 6   age             159256 non-null  int64
 7   hearing(left)   159256 non-null  int64
 8   waist(cm)       159256 non-null  float64
 9   hemoglobin      159256 non-null  float64
 10  height(cm)      159256 non-null  int64
 11  smoking         159256 non-null  int64
dtypes: float64(2), int64(10)
memory usage: 14.6 MB
```

[16]:
```
data.isna().sum()
```

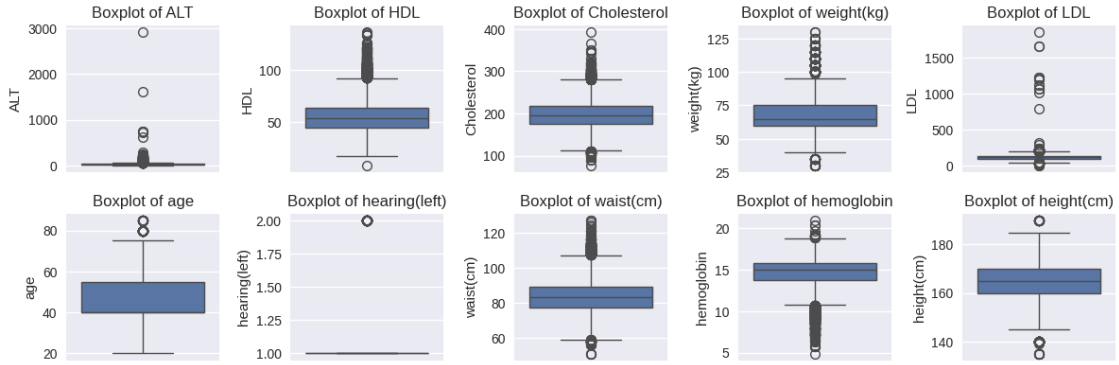
[16]:
```
id              0
ALT             0
HDL             0
Cholesterol     0
weight(kg)      0
LDL             0
age             0
hearing(left)   0
waist(cm)       0
hemoglobin      0
height(cm)      0
smoking         0
dtype: int64
```

[17]:
```
numerical_cols = ['ALT', 'HDL', 'Cholesterol', 'weight(kg)', 'LDL', 'age',␣
 ↪'hearing(left)', 'waist(cm)', 'hemoglobin', 'height(cm)']

plt.figure(figsize=(12, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(5, 5, i)
    sns.boxplot(y=data[col])
```

```
    plt.title(f'Boxplot of {col}')
    plt.tight_layout()

plt.show()
```



[18]:
```
for col in data.select_dtypes(include=[np.number]).columns:
    if col =='dental caries':
        continue
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    upper_bound = Q3 + 1.8 * IQR
    lower_bound = Q1 - 1.5 * IQR
    outliers_below = data[data[col] < lower_bound].shape[0]
    outliers_above = data[data[col] > upper_bound].shape[0]
    if outliers_below + outliers_above > 500:
        # handle values above the upper bound
        data[col] = np.where(data[col] > upper_bound, upper_bound, data[col])
        # handle values below the lower bound
        data[col] = np.where(data[col] < lower_bound, lower_bound, data[col])
    else :
        upper_bound = Q3 + 1.6 * IQR
        lower_bound = Q1 - IQR
        data[col] = np.where(data[col] > upper_bound, upper_bound, data[col])
        data[col] = np.where(data[col] < lower_bound, lower_bound, data[col])
```

[19]:
```
data.describe()
```

[19]:

	id	ALT	HDL	Cholesterol	\
count	159256.000000	159256.000000	159256.000000	159256.000000	
mean	79627.500000	25.923458	55.826788	195.835558	
std	45973.391572	12.916673	13.872204	28.163075	
min	0.000000	1.000000	16.500000	133.000000	

```
25%     39813.750000      16.000000      45.000000     175.000000
50%     79627.500000      22.000000      54.000000     196.000000
75%    119441.250000      32.000000      64.000000     217.000000
max    159255.000000      60.800000      98.200000     284.200000

          weight(kg)            LDL            age  hearing(left)  \
count  159256.000000  159256.000000  159256.000000       159256.0
mean       67.113562     114.546457      44.421366            1.0
std        12.483615      25.761897      11.601336            0.0
min        37.500000      57.000000      25.000000            1.0
25%        60.000000      95.000000      40.000000            1.0
50%        65.000000     114.000000      40.000000            1.0
75%        75.000000     133.000000      55.000000            1.0
max       102.000000     193.800000      79.000000            1.0

           waist(cm)     hemoglobin     height(cm)        smoking
count  159256.000000  159256.000000  159256.000000  159256.000000
mean       83.034845      14.807563     165.378290       0.437365
std         8.822533       1.393455       8.577791       0.496063
min        65.000000      10.800000     150.000000       0.000000
25%        77.000000      13.800000     160.000000       0.000000
50%        83.000000      15.000000     165.000000       0.000000
75%        89.000000      15.800000     170.000000       1.000000
max       108.200000      19.400000     186.000000       1.000000
```

[20]:
```python
X = data.drop(['id', 'smoking'], axis=1)
y = data['smoking']
print("Features shape:", X.shape)
print("Target shape:", y.shape)
```

```
Features shape: (159256, 10)
Target shape: (159256,)
```

1.2 Data split

[21]:
```python
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.4,
    random_state=42,
    stratify=y
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5,
    random_state=42,
    stratify=y_temp
```

```
)

print("Train shape:", X_train.shape, y_train.shape)
print("Validation shape:", X_val.shape, y_val.shape)
print("Test shape:", X_test.shape, y_test.shape)
```

```
Train shape: (95553, 10) (95553,)
Validation shape: (31851, 10) (31851,)
Test shape: (31852, 10) (31852,)
```

1.3 Main model

[23]:
```
def bagging(X_train, y_train, X_test, n_estimators=250, max_samples_ratio=0.4,␣
↪random_state=42):
    np.random.seed(random_state)

    X_train_np = X_train.values if isinstance(X_train, pd.DataFrame) else␣
↪X_train
    y_train_np = y_train.values if isinstance(y_train, pd.Series) else y_train
    X_test_np  = X_test.values if isinstance(X_test, pd.DataFrame) else X_test

    n_samples = X_train_np.shape[0]
    max_samples = int(max_samples_ratio * n_samples)

    estimators = []

    for i in range(n_estimators):
        indices = np.random.choice(n_samples, size=max_samples, replace=True)
        X_bootstrap = X_train_np[indices]
        y_bootstrap = y_train_np[indices]

        tree = DecisionTreeClassifier(random_state=random_state + i)
        tree.fit(X_bootstrap, y_bootstrap)
        estimators.append(tree)

    preds = []
    for tree in estimators:
        p = tree.predict(X_test_np)
        preds.append(p)

    preds = np.array(preds).T

    final_predictions = []
    for row in preds:
        vals, counts = np.unique(row, return_counts=True)
        majority_vote = vals[np.argmax(counts)]
        final_predictions.append(majority_vote)
```

```
    return np.array(final_predictions)

y_pred_bag = bagging(X_train, y_train, X_test)
acc_bag = accuracy_score(y_test, y_pred_bag)

print("Accuracy:", acc_bag)
print("Classification Report:\n", classification_report(y_test, y_pred_bag))
```

```
=== Bagging (from scratch) ===
Accuracy: 0.7478337310059023
Classification Report:
               precision    recall  f1-score   support

         0.0       0.82      0.70      0.76     17921
         1.0       0.68      0.80      0.74     13931

    accuracy                           0.75     31852
   macro avg       0.75      0.75      0.75     31852
weighted avg       0.76      0.75      0.75     31852
```