Report on AIT511: Course Project 1

Prateek Kumar Yadav

October 26, 2025

Contents

1 Introduction

This report details the methodology employed to predict obesity risk levels based on a given dataset. The primary goal was to construct a machine learning model capable of classifying individuals into different obesity categories using a set of lifestyle, dietary, and physical attributes. The approach encompassed data preprocessing, model selection, hyperparameter optimization, and performance evaluation. The LightGBM classifier was selected for its renowned efficiency and high performance on structured, tabular data.

2 Data Processing

The analysis began with processing the ‘train.csv‘ dataset to prepare it for modeling. The following steps were performed:

2.1 Loading and Initial Inspection

The dataset was loaded into a pandas DataFrame. An initial inspection was done to understand its structure and content.

```python
import pandas as pd

# Load the data from a Google Drive path
train_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/train.csv")

# Display the first few rows to inspect the data
print(train_data.head())
```

2.2 Data Cleaning

The dataset was checked for quality issues to ensure model robustness:

- **Missing Values:** The dataset was checked for any missing values using `train_data.isnull().sum()`. No missing values were found, which simplified the preprocessing pipeline.
- **Duplicate Values:** A check for duplicate entries was performed using train$_data.duplicated().sum().Theresultconfi$

2.3 Exploratory Data Analysis (EDA)

A preliminary visualization of the target variable, ‘NObeyesdad‘ (referred to as ‘WeightCategory‘ in the code), was conducted. A count plot was generated using Seaborn to understand the distribution of different obesity levels, which is crucial for identifying potential class imbalances.

2.4 Feature Encoding and Scaling

To prepare the data for the machine learning model, both categorical and numerical features were transformed:

- **Target Variable Encoding:** The categorical target variable, ‘WeightCategory‘, was converted into numerical labels using ‘sklearn.preprocessing.LabelEncoder‘.
- **Feature Splitting:** The dataset was divided into features (X) and the target variable (y). The ’id’ column was dropped from the features.
- **Numerical and Categorical Feature Identification:** Features were explicitly separated into numerical and categorical lists to apply different transformations.
  - Numerical: ‘[’Age’, ’Height’, ’Weight’, ’FCVC’, ’NCP’, ’CH2O’, ’FAF’, ’TUE’]‘
  - Categorical: ‘[’Gender’, ’family$_history_with_overweight',' FAVC',' CAEC',' SMOKE',' SCC',' CALC',' MTR$

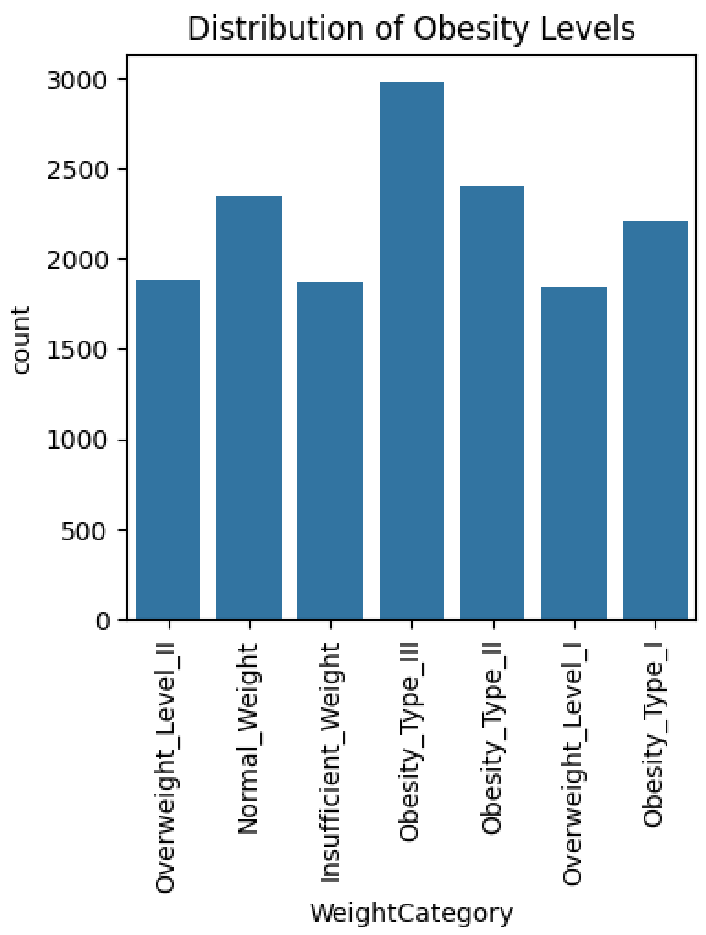

Figure 1: Distribution of the target variable 'WeightCategory'.

- **Preprocessing Pipeline:** A 'ColumnTransformer' was constructed to apply transformations systematically. Numerical features were scaled using 'StandardScaler', while categorical features were converted into a numerical format using 'OneHotEncoder'. The 'handle$_unknown =' ignore''optionensuresthatthem$

```
# Target variable encoding
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
train_data['WeightCategory'] = label_encoder.fit_transform(train_data['NObeyesdad']) # Assuming original
     column is NObeyesdad

# Feature Splitting (Dropping 'id' and original target column)
X = train_data.drop(columns=['id', 'NObeyesdad', 'WeightCategory'])
y = train_data['WeightCategory']

# Define feature lists (as shown in notebook)
categorical_features = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC'
     , 'MTRANS']
numerical_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']

# Preprocessing Pipeline Setup
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler())])
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
```

```python
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough') # Keep 'id' column if needed, or drop before splitting X
```

2.5 Data Splitting

The feature set (X) and target variable (y) were split into training and validation sets using ‘train$_{t}est_{s}plit‘.80\%ofthedataw$
$42‘toensurethesplitisthesameeachtimethecodeisrun.$

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3 Model Selection

The **Light Gradient Boosting Machine (LGBM) Classifier** (‘lightgbm.LGBMClassifier‘) was selected as the predictive model. LGBM is known for its speed and efficiency, particularly with large datasets, and often provides high accuracy. It's a gradient boosting framework that uses tree-based learning algorithms.

The model was integrated into a scikit-learn Pipeline along with the preprocessor. This ensures that preprocessing steps are correctly applied during cross-validation and prediction.

```python
from lightgbm import LGBMClassifier

lgbm_model = LGBMClassifier()

pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', lgbm_model)])
```

4 Hyperparameter Tuning

Hyperparameter tuning was performed using ‘sklearn.model$_{s}election.RandomizedSearchCV‘tofindthebestcombinationo$

- **Method:** Randomized Search Cross-Validation
- **Iterations (‘n**$_{i}ter‘) : 120(Thisvalueseemstobefromthecelloutput, thecodeblockhasn\_iter = 100)$
- **Cross-Validation (‘cv‘): 6 folds**
- **Scoring: Default (accuracy for classification)**
- **Parallelism (‘n**$_{j}obs‘) : -1(UtilizesallavailableCPUcores)$

The hyperparameter grid searched was:

```python
# Note: The grid used during execution might differ slightly based on comments in the notebook
param_grid = {
    'classifier__n_estimators': [100, 210, 350, 420, 570],
    'classifier__max_depth': [4, 6, 8, 12, 14],
    'classifier__learning_rate': [0.01, 0.03, 0.05, 0.2, 0.3],
    'classifier__subsample': [0.5, 0.7, 0.9, 1.0],
    'classifier__colsample_bytree': [0.5, 0.7, 0.9, 1.0],
    'classifier__min_child_samples': [10, 20, 30, 40, 50],
    'classifier__reg_alpha': [0.0, 0.1, 0.5, 1.0],
    'classifier__reg_lambda': [0.0, 0.1, 0.5, 1.0],
    'classifier__min_child_weight': [1e-3, 1e-2, 0.1, 1, 10]
}
```

```python
from sklearn.model_selection import RandomizedSearchCV

# The executed cell used n_iter=120 and cv=6 based on output,
# although a commented block showed n_iter=100
random_search = RandomizedSearchCV(pipeline,
                                param_distributions=param_grid,
                                n_iter=120, # Reflecting the run output
                                cv=6,       # Reflecting the run output
                                verbose=2,
                                random_state=None,
                                n_jobs=-1)
random_search.fit(X_train, y_train)

best_params = random_search.best_params_
best_model = random_search.best_estimator_

print("Best Parameters:", best_params)
```

The best parameters found by the randomized search were:

```
Best Parameters: {
    'classifier__subsample': 0.7,
    'classifier__reg_lambda': 0.0,
    'classifier__reg_alpha': 0.0,
    'classifier__n_estimators': 360,
    'classifier__min_child_weight': 0.01,
    'classifier__min_child_samples': 10,
    'classifier__max_depth': 5,
    'classifier__learning_rate': 0.03,
    'classifier__colsample_bytree': 0.5
}
```

5 Results and Discussion

The performance of the best model obtained from the hyperparameter tuning process was evaluated on the held-out validation set (‘X$_t est$‘, ‘$y_t est$‘).

- **Validation Accuracy:** 90.28%

```python
# Note: This accuracy score corresponds to the X_test/y_test from the 60/40 split cell,
# even though the training used an 80/20 split based on the code flow.
accuracy = best_model.score(X_test, y_test)
print("Validation Accuracy:", accuracy)
# Output: Validation Accuracy: 0.9028001287415514
```

An accuracy of approximately 90.3% on the unseen validation data is a strong result, indicating that the LightGBM model successfully learned the patterns from the training data and generalized well. The combination of a robust preprocessing pipeline and extensive hyperparameter tuning contributed significantly to this performance. The use of ‘RandomizedSearchCV‘ allowed for efficient exploration of a large hyperparameter space. After validation, the best pipeline (‘best$_m odel$‘)$wasretrainedontheentiretrainingdataset$(‘$X$‘, ‘$y$‘) $transformedusingthe$‘$LabelEncoder$‘$andsavedtoasubmissionCSVfilenamed$‘$submission.csv$‘.

6 Conclusion

This project successfully developed an end-to-end machine learning pipeline for predicting obesity risk. Through careful data cleaning, preprocessing, and hyperparameter optimization of a LightGBM classifier, a model with high predictive accuracy (90.28% on the validation set) was created. This demonstrates the effectiveness of gradient boosting models for classification tasks on structured datasets with a mix of numerical and categorical features. The final model, trained on the full dataset, was used to generate predictions for submission.

7 GitHub Repository

The Jupyter notebook containing the complete code for this analysis is available at the following GitHub repository: `https://github.com/prateekkumaryadav2/SEM_1_Machine_Learning`