CA1 Assignment 1: ML Foundations

Roland Oteniya: 201884707
Abdulla Khrais: 201874468
Mohamed Mohamed: 201422688
Jon Rajabi: 201912133

Introduction

This project's goal is to build a predictive machine learning model for a medical organisation specialising in obesity treatment that can predict patients' obesity levels based on different parameters.

In this project, we followed a systematic data science pipeline. In the data management phase, we analysed, cleaned and divided the data using an 80:20 split for training and testing. Secondly, since this is a classification project and we have labelled data, we decided to use supervised learning models. Finally, for the evaluation stage, we ran the models without tuning to establish baseline metrics and then performed a stratified 5-fold cross-validation hyperparameter tuning technique. After testing the final models on precision, accuracy and recall, we found that Support Vector Machine was the best model on the unseen data.

Data Management

Problem Modelling

We modelled this case study as a Supervised multi-class classification problem, as the ground truth 'obesity level' consists of 7 distinct categories ranging from 'Insufficient weight' to 'Obesity Type III'. Supervised learning techniques were selected as we've been given previous patients' labelled data.

Data Analysis

1. By using the 'isnull' function, we found no missing or NAN values.
2. By plotting a correlation matrix on all the numerical parameters of the dataset, we found that FD (Fruit portion per day) and Meals (Meals per day) were over 93% correlated. To prevent multicollinearity and remove redundant information for our models, we dropped FD for our testing.

Figure 1: Correlation Matrix for Numerical Features.

3. To identify outliers, we produced box plots for all numerical parameters. We identified 3 outliers, which were removed to avoid negatively affecting the training of the models.

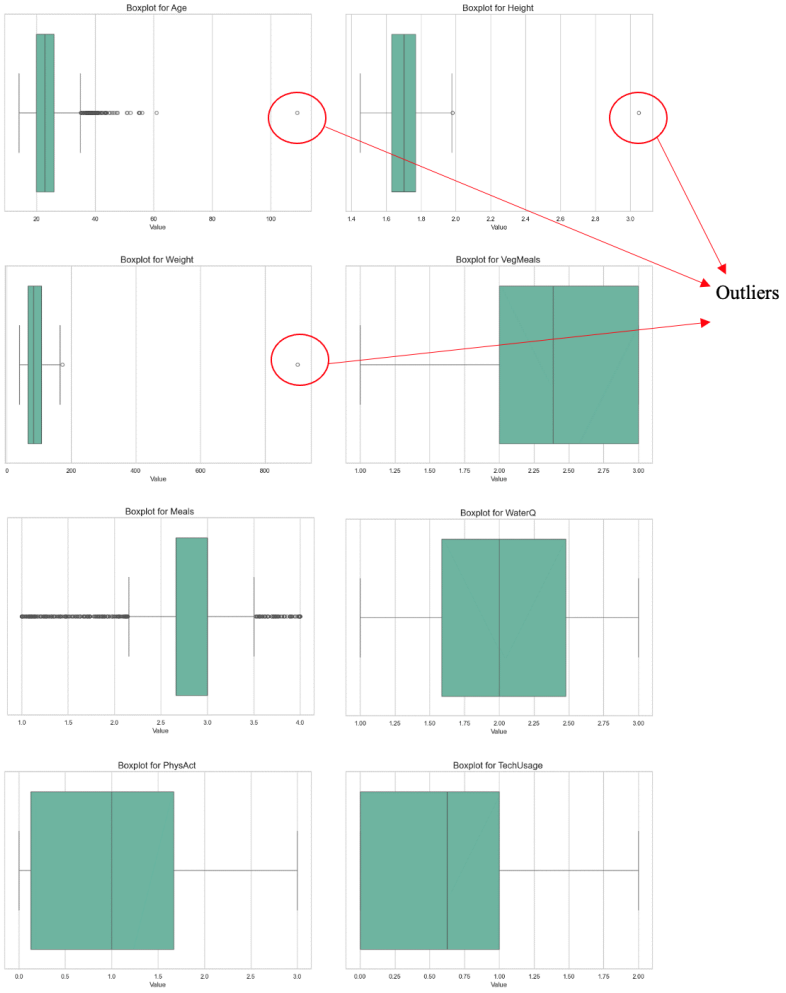

Figure 2: Boxplot for Numerical Features

4. To assess class distribution, we plotted a bar chart of the target variable 'ObesityLvl'. This ensured no significant class imbalance.

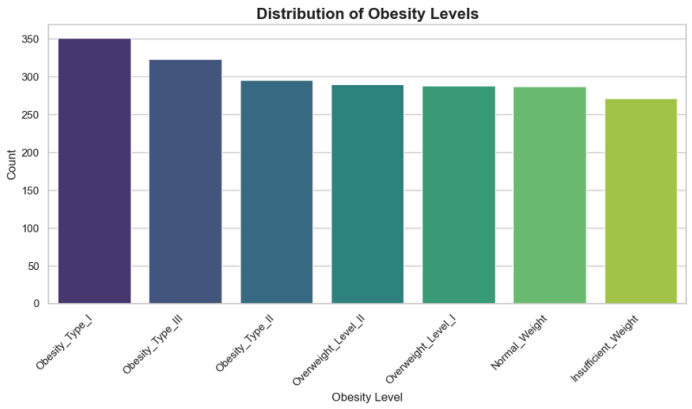

Figure 3: Bar chart of the target variable.

Data Pre-Processing

To make the data ready to fit onto the models, we preprocessed it by encoding and scaling it.

- Encoding: Ordinal, non-numerical features (MealsBetw and AlcConsume) were label encoded, while Categorical features (Gender, family_overweight and TransMeans) were transformed using One-Hot encoding with the 'drop_first=True' technique to avoid multicollinearity.
- Scaling: Continuous features were standardised using the 'StandardScaler' technique to prevent magnitude bias in distance-based algorithms like SVM (Support Vector Machines) and KNN (K-Nearest Neighbour).

Experimental Protocol

We used an 80:20 split to divide the dataset into training and testing sets using stratified sampling to maintain class proportions across all splits. To validate model performance and optimise hyperparameters, we used a stratified 5-fold cross-validation on the training set. A fixed random seed (random_state=42) was used to give replicable results. As the analysis showed no signs of unbalanced class distribution, we didn't use any resampling techniques like SMOTE or class weighting adjustments.

Model Training

Initial Model Evaluation

To establish a performance baseline, we evaluated 6 different Machine Learning techniques. We used stratified cross-validation for this assessment so that each validation fold properly represented the ground truth's distribution.

The ML models used were:

1. Decision Tree: A tree built by recursive splits that minimises Gini impurity or maximises gain. It's simple but prone to overfitting.

2. Random Forest: An ensemble of decision trees trained on random data subsets (bagging). It uses a majority vote for the final class, reducing variance.
3. Support Vector Machine (SVM): Finds the optimal hyperplane (decision boundary) by creating the maximum margin between classes. It uses the kernel trick for non-linear data.
4. K-Nearest Neighbours (KNN): Classifies a new sample using a majority vote from its k-nearest neighbours. It has no training step.
5. Naive Bayes: A probabilistic classifier based on Bayes' Theorem. It "naively" assumes all features are conditionally independent.
6. Logistic regression: Adapts linear regression for classification by passing the result through a sigmoid function to model class probability.

After reviewing the initial baseline results (Figure4), we decided to remove the Naïve Bayes model from further testing and tuning. With an accuracy of 0.54 and F1-Score of 0.46, we concluded that since its performance was significantly lower than the other models, which averaged 0.89 accuracy and an F1-Score of 0.89, it would not be suitable for this problem.

Model	Metric	Score
Decision Tree	Accuracy	0.9241
	F1-Score	0.9237
	Recall	0.9241
	Precision	0.9252
Random Forest	Accuracy	0.9484
	F1-Score	0.9488
	Recall	0.9484
	Precision	0.9513
SVM	Accuracy	0.9116
	F1-Score	0.9123
	Recall	0.9116
	Precision	0.9149
KNN	Accuracy	0.8114
	F1-Score	0.8006
	Recall	0.8114
	Precision	0.8101
Naive Bayes	Accuracy	0.5409
	F1-Score	0.46
	Recall	0.5409
	Precision	0.5063
Logistic Regression	Accuracy	0.8737
	F1-Score	0.8716
	Recall	0.8737
	Precision	0.8754

Figure 4: Model with baseline scores

Hyperparameter Tuning

Following the baseline testing, we tuned the hyperparameters. The 5 chosen models, Decision Tree, Random Forest, SVM, KNN and Logistic Regression, were tuned to find their best performing configurations.

Methodology

For Hyperparameter Tuning, we used GridSearchCV. This technique tests every combination of given hyperparameters using cross-validation and picks the best performing one. Although there are more complex techniques like Bayesian Optimisation, we used GridSearchCV as it is the best option for this scenario, where we want replicable results (GridSearchCV is deterministic, unlike Bayesian optimisation), and we want to guarantee that all parameter combinations are tested.

Model	Hyperparameters		Best Parameters	Definition
Decision Tree	Max Depth	[5,10,20,30,none]	10	Maximum number of levels/splits in the tree; lir growth to prevent overfitting.
	min_samples_split	[2, 5, 10, 20]	2	Minimum number of samples required to split a node; higher values prevent overfitting.
	min_samples_leaf	[1, 2, 4, 8]	2	Minimum number of samples required in a leaf ensures leaves represent meaningful patterns.
	criterion	['gini', 'entropy']	entropy	Impurity measure used for splitting ('gini' for G impurity, 'entropy' for information gain).
Random Forest	n_estimators	[50, 100, 200]	200	Number of decision trees in the forest; more tr generally improve performance but increase computation.
	max_depth	[10, 20, 30, None]	20	Maximum depth of each tree in the forest.
	min_samples_split	[2, 5, 10]	2	Minimum samples needed to split nodes in eac
	min_samples_leaf	[1, 2, 4]	1	Minimum samples required in leaf nodes of eac
SVM	C	[0.1, 1, 10, 100]	10	Regularization parameter; controls trade-off be maximizing margin and minimizing classificatior (smaller C = wider margin, more tolerance for
	kernel	['linear', 'rbf', 'poly']	linear	Transformation function ('linear', 'rbf', 'poly') th data to higher dimensions for non-linear separa
	gamma	['scale', 'auto']	scale	Defines influence range of single training examp 'rbf' and 'poly' kernels ('scale' uses 1/(n_featur variance), 'auto' uses 1/n_features).
KNN	n_neighbors	[3, 5, 7, 9, 11, 15, 20]	3	Number of nearest neighbors (k) to consider classification; affects decision boundary smoot
	weights	['uniform', 'distance']	distance	Weighting scheme ('uniform' treats all neighbor 'distance' weights closer neighbors more heavi
	metric	['euclidean', 'manhattan', 'minkowski']	manhattan	Distance measure ('euclidean', 'manhattan', 'mi used to determine nearest neighbors.
Logistic Regression	C	[0.01, 0.1, 1, 10, 100]	10	Inverse regularization strength; smaller values s stronger regularization to prevent overfitting.
	penalty	['l1', 'l2']	l1	Regularization type ('l1' for Lasso, 'l2' for Ridg controls coefficient shrinkage.
	solver	['liblinear', 'saga']	saga	Optimization algorithm ('liblinear' for small data 'saga' for large datasets and all penalties).
	max_iter	[1000, 2000]	2000	Maximum number of iterations for the solver to increase if convergence warnings appear.

Figure 5: Models and Hyperparameters

Evaluation

After an extensive hyperparameter tuning phase, the resulting model accuracies are presented in Figure 1. Although Logistic Regression showed the most improvement after tuning of 0.082, it wasn't the outright best performing model, which was SVM with a Tuned Accuracy of 0.96144. As a result, it was selected to be tested on the final unseen data.

Model	Baseline Accuracy	Tuned Accuracy	Improvment
SVM	0.911636	0.96144	0.049804
Logistic Regression	0.873671	0.955512	0.081841
Random Forest	0.948401	0.953744	0.005343
Descision Tree	0.924081	0.940101	0.01602
KNN	0.811404	0.881973	0.070569

Figure 6: Table of model results pre and post-hyperparameter tuning.

On the Unseen test set, SVM still performed impressively well with an accuracy score of 0.959716. This was within 0.2 of the 0.96 that the model scored in accuracy on the 5-fold cross-validation, which is strong evidence that the model generalised well and didn't overfit on the training data.

To understand the final SVM's performance in detail, we looked at 3 key metrics:

1. Accuracy: 95.97%. The model correctly classified 405 out of 422 samples in the test set.
2. Weighted F1-Score: 95.96%. This metric takes the average F1-score across all classes, weighted by the number of true instances in each class.
3. Macro F1-Score: 95.92%. This metric is the unweighted average F1-score across all classes, giving each class equal importance.

The fact that the Weighted and Macro F1-Scores are almost identical is an important finding. It proves that the model's high performance is robust, and it accurately classifies all categories from 'Insufficient_Weight' to 'Obesity_Type_III'.

Confusion Matrix Analysis

To get a deeper understanding of the model's decision-making making we created and analysed a confusion matrix (Figure 7).

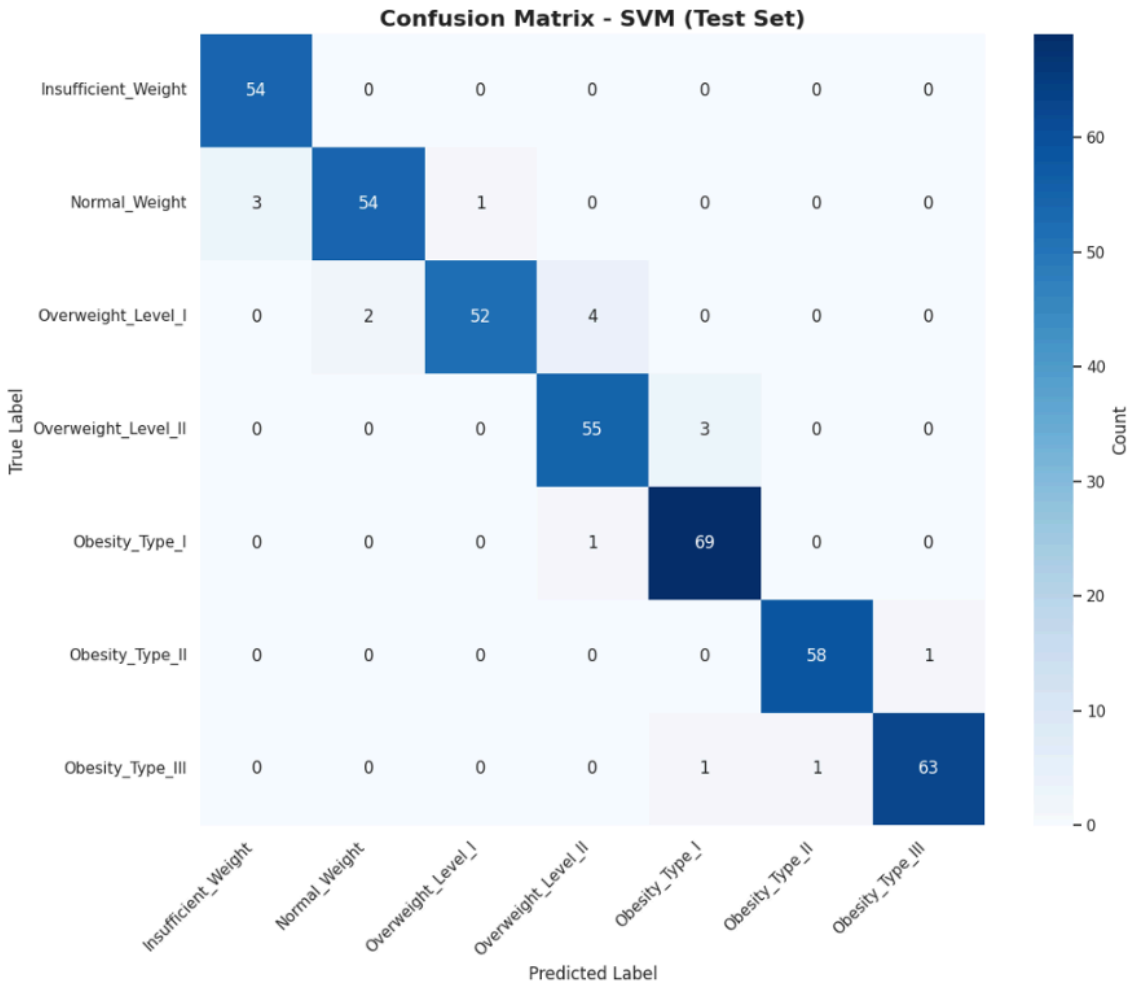

Figure 7: Confusion Matrix for final SVM's classifications

The matrix shows a strong diagonal, showing that a high number of correct predictions were made. Out of 422 cases, only 17 were misclassified (4.03%).
Analysing the misclassifications further, we found that the errors are not random; they are almost always between adjacent categories. For example, 4 samples of 'Overweight_Level_I' were mistaken for 'Overweight_Level_II'. These mistakes are understandable as the feature boundaries between classes may be marginal.

There was only 1 occasion in which the error was more than 1 adjacent category. A true label of ‘Obesity_Type_III’ was misclassified as ‘Obesity_Type_I’. Other than this instance, the model did not make any severe errors. This demonstrates high reliability, making it an effective tool for the medical institution.