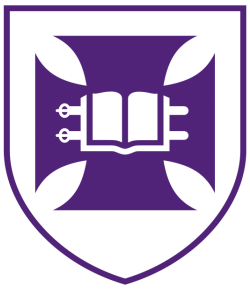
THE UNIVERSITY OF QUEENSLAND
FACULTY OF ENGINEERING, ARCHITECHTURE AND INFORMATION TECHNOLOGY



THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

COMP4702 ASSIGNMENT REPORT

MACHINE LEARNING

Submitted by:
Nguyen Van Quoc Chuong – 47787810

Brisbane city, 5/2023

Contents

1 Introduction

The provided data encompasses various nutritional indicators for different types of food and drink, including calcium, energy, and more. The dataset consists of 6 core nutrition categories: proximates, vitamins, minerals, fatty acids, amino acids, and others. Each core nutrition category comprises multiple columns representing specific nutrients. For the purpose of this study, the measurement of fatty acids nutrition will be based on grams rather than the

In this report, I will employ various machine learning algorithms such as KNN, SVM, Random Forest, Logistic Regression, and ANN to perform binary classification, specifically distinguishing between liquid and solid forms of food. The objective is to predict the classification based on the given nutritional indicators.

Additionally, I will address the issue of imbalanced class distribution within the dataset by utilizing the K-means algorithm. This approach aims to generate new classes, subsequently allowing for a more even distribution of data among the classes. The original class classification labels will be used to assess the new distribution within the updated classification labels. Moreover, I will apply PCA (Principal Component Analysis) to effectively reduce the number of features while providing a sound justification for the selected reduction.

Overall, this report will explore the application of machine learning techniques to predict food classification and address class imbalance concerns, ultimately striving to enhance the understanding of the dataset's nutritional characteristics.

2 Binary classification problem

2.1 Preprocessing

2.1.1 Remove null values

The dataset contained numerous null values in the features columns, except for the "Classification," "Key," and "Food Name" columns. After careful consideration, it was decided to eliminate these columns from the dataset. Instead, the focus was placed on retaining the main columns related to the core nutrition categories specified in the provided sub-dataset. The deleted columns were found to be granular and extensive, but due to the substantial presence of null values, it was deemed more appropriate to handle these missing values through techniques like mean or median imputation, or filling them with a specific number. It is worth noting that the deleted columns may not significantly impact future predictions.

Additionally, an important step in the preprocessing phase involved dimensionality reduction. By reducing the number of features, the dataset was streamlined to include 64 essential features. This reduction in dimensions mitigates the risk of the "curse of dimensionality," which refers to the problem encountered when the number of dimensions exceeds the available observations. In such cases, the data points become sparse and are no longer closely located to one another, posing challenges for algorithms like KNN (K-Nearest Neighbors).

By reducing the number of features, the dataset becomes more manageable and alleviates potential issues related to high-dimensional data. This enhancement ensures the feasibility of implementing the KNN algorithm and enhances the overall effectiveness of the subsequent analysis.

2.1.2 Labeling for binary classification problem

In order to assign labels for the binary classification problem, I performed a matching process using the 'Public Food Key' column from the 'Liquids only per 100mL' sheet in the Excel data. This involved comparing the values from this column with the corresponding 'Public Food Key' column in the 'All solids liquids per 100g' sheet.

The objective was to label each food item based on its form, either liquid or non-liquid (solid). To accomplish this, a new column called 'is_liquid' was created. If the food ID in the liquid sheet matched with a corresponding ID in the 'All solids liquids per 100g' sheet, the value of 'is_liquid' was marked as 1, indicating that the food item is in liquid form. Conversely, a value of 0 was assigned to indicate that the food item is not in liquid form.

By performing this labeling process, each food item in the dataset was associated with a binary classification label, facilitating the subsequent analysis and modeling tasks.

2.1.3 Splitting train, test sets and normalization

In the preprocessing phase of the project, the following steps were performed to prepare the data for training and testing the binary classification model.

2.1.3.a Feature Selection:

The feature matrix, denoted as 'X', was constructed by selecting the columns from the 'all_df' dataframe, excluding the 'Public Food Key', 'Classification', and 'Food Name' columns. This selection was achieved using the .iloc[:, 3:-1] indexing, capturing all rows and columns starting from the 4th column up to the second-to-last column.

2.1.3.b Target Variable Extraction:

The target variable, 'y', representing the 'is_liquid' column, was extracted using the .iloc[:, -1] indexing, which includes all rows and only the last column of the 'all_df' dataframe.

2.1.3.c Train-Test Split:

The dataset was divided into training and testing sets using the 'train_test_split' function from the scikit-learn library. The data was split into X_train, X_test, y_train, and y_test sets. The test set was allocated 30% of the data, while the remaining 70% was used for training. The 'shuffle=True' parameter ensured random shuffling of the data before the split, and 'random_state=42' was set to maintain reproducibility of the split.

2.1.3.d Data Scaling:

To ensure that all features are on a similar scale, a MinMaxScaler from the scikit-learn library was utilized. The scaler was initialized as 'scaler = MinMaxScaler()', and the training data, X_train, was transformed using 'X_train = scaler.fit_transform(X_train)'. This step computes the minimum and maximum values of each feature in the training set and scales the data accordingly. Subsequently, the same scaling transformation was applied to the testing data using 'X_test = scaler.transform(X_test)'.

By following these steps, the data was appropriately preprocessed, and the feature matrix, X, and target variable, y, were prepared for further analysis and modeling. The train-test split facilitated model training on the training set and evaluation on the independent testing set, ensuring unbiased performance assessment. Scaling the data using the MinMaxScaler maintained the relative relationships between the features while placing them within a consistent range, supporting the training and convergence of machine learning models.

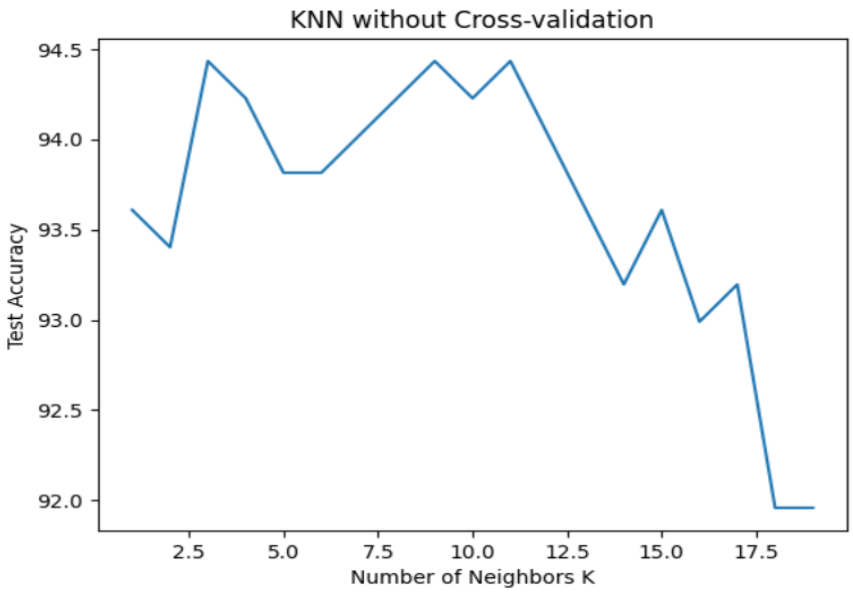
2.2 Models' implementations

2.2.1 K-Nearest Neighbors

In order to determine the optimal number of neighbors (k) for the KNN (K-Nearest Neighbors) algorithm, two approaches were implemented and evaluated: one without cross-validation and another with cross-validation.

2.2.1.a KNN without Cross-Validation:

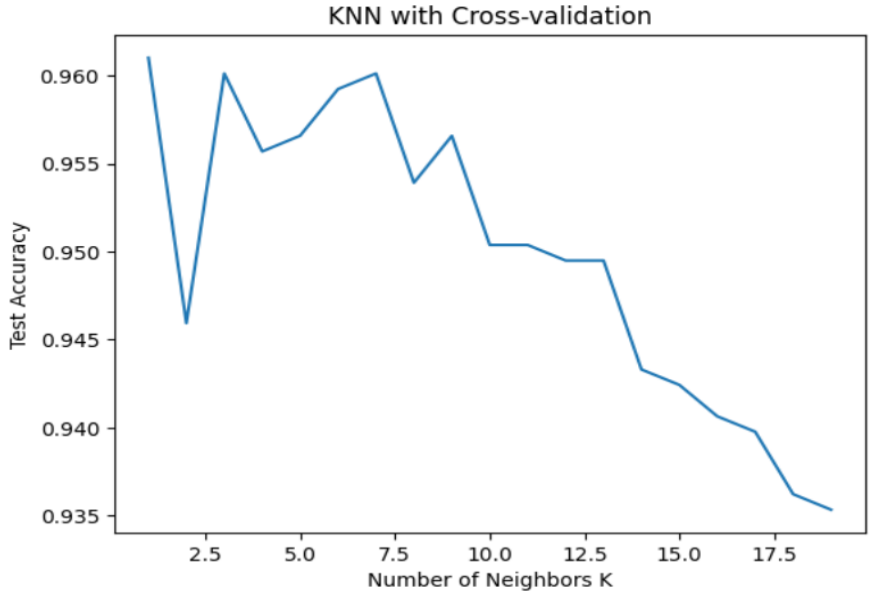
Initially, a loop was used to iterate over a range of k values from 1 to 19. For each k value, a KNeighborsClassifier object was instantiated with the specified number of neighbors. The model was then trained on the training set (X_train and y_train). Subsequently, predictions were made on the testing set (X_test), and the accuracy of the predictions was computed using the accuracy_score function. The accuracy was stored in a list, along with the corresponding k value. Finally, the k value associated with the highest accuracy was identified, and the optimal number of neighbors was determined.



2.2.1.b KNN with Cross-Validation:

Similarly, a loop was employed to iterate over the same range of k values from 1 to 19. For each k value, a KNeighborsClassifier object was created, and cross-validation was performed

using the cross_val_score function. The 'cv=10' parameter specified a 10-fold cross-validation, dividing the training set into 10 equal-sized subsets. The mean accuracy across all folds was calculated and stored in a list along with the corresponding k value. Again, the k value associated with the highest mean accuracy was identified, yielding the optimal number of neighbors.



Both methods, with and without cross-validation, consistently indicated that choosing k = 1 for the KNN algorithm resulted in the highest accuracy rates of 94.43% and 96.10%, respectively. These findings demonstrate the effectiveness of relying on the nearest neighbor for accurate binary classification of food items as liquid or solid.

2.2.2 Artificial neural network

For the implementation of the Artificial Neural Network (ANN), the following steps were taken:

2.2.2.a Early Stopping:

Early stopping was applied using the EarlyStopping callback with a patience of 5. This helps prevent overfitting by stopping the training process if the validation loss does not improve after a certain number of epochs.

2.2.2.b Model Architecture:

The ANN model was constructed using the Sequential API. The architecture consists of multiple dense (fully connected) layers. The input layer has 16 units with the 'relu' activation function, which takes the shape of the input data (X_train.shape[1]). The subsequent layers include 8, 6, and 4 units, respectively, all activated using the 'relu' function. The output layer consists of a single unit activated by the 'sigmoid' function for binary classification.

2.2.2.c Compilation:

The model was compiled using the 'adam' optimizer, 'binary_crossentropy' as the loss function for binary classification, and 'accuracy' as the evaluation metric.

2.2.2.d Training:

The model was trained on the training data (X_train and y_train) for a maximum of 500 epochs, with a batch size of 32. A validation split of 0.2 was used to allocate a portion of the training data for validation during the training process. The EarlyStopping callback was utilized to monitor the validation loss and stop training if it does not improve within the specified patience. The training progress was displayed with verbosity set to 1.

2.2.2.e Evaluation:

After training, the model's performance was evaluated using the test data (X_test and y_test). The test loss and accuracy were computed using the 'evaluate' method of the trained model and printed to the console for analysis.

In python code, my ANN was implemented as follow:

```
early_stopping_monitor = EarlyStopping(patience=5)
ann = Sequential()
ann.add(Dense(16, activation='relu', input_dim=X_train.shape[1]))
ann.add(Dense(8, activation='relu'))
ann.add(Dense(6, activation='relu'))
ann.add(Dense(4, activation='relu'))
ann.add(Dense(1, activation='sigmoid'))

ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

ann_history = ann.fit(X_train, y_train, epochs=500,
       batch_size=32,
       validation_split = 0.2,
       callbacks =[early_stopping_monitor],
       verbose=1)

loss, accuracy = ann.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

The summary for my ANN:

To calculate the number of parameters in each layer of a dense (fully connected) neural network, I can use the following formula:

$$Params = (input\_dim + 1) * output\_dim$$

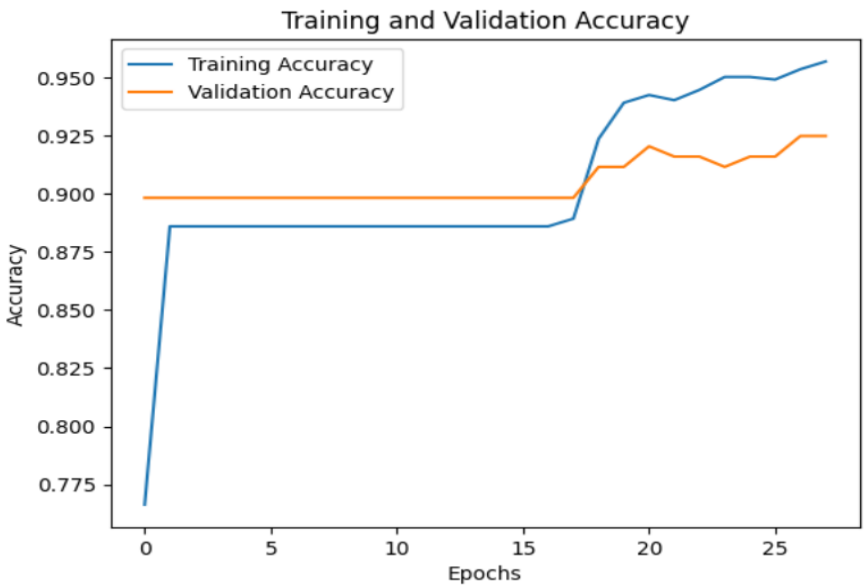
In this formula:

- $input\_dim$ represents the number of units in the previous layer.
- $output\_dim$ represents the number of units in the current layer.
- The "+1" term accounts for the bias term associated with each neuron.

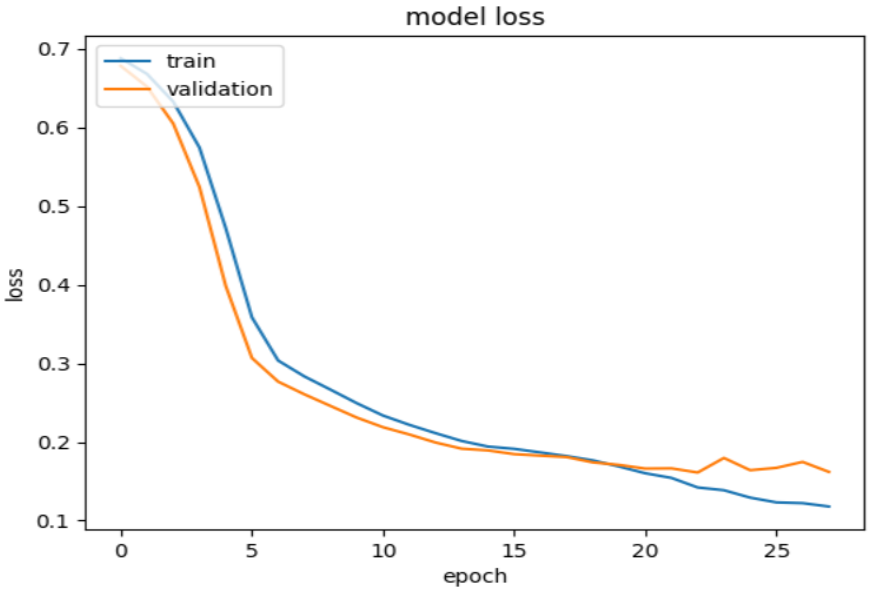
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                976

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 6)                 54

 dense_3 (Dense)             (None, 4)                 28

 dense_4 (Dense)             (None, 1)                 5

=================================================================
Total params: 1,199
Trainable params: 1,199
Non-trainable params: 0
```

Let's take Layer 1 as an example:
Layer 1 (Dense):
- Input Dimension: $X_{\text{train.shape[1]}}$ (number of features in the input data, there are 60 features)
- Output Dimension: 16
- Params = $(X_{\text{train.shape[1]}} + 1) \times 16 = (60 + 1) \times 16 = 976$

Result:



The loss and accuracy figures indicate that the ANN model starts to overfit after epoch 20. To prevent further overfitting, an early stopping mechanism was implemented, where if the accuracy did not improve after 5 epochs, the training process was stopped.



2.2.3 Support Vector Machine

In order to assess the performance of Support Vector Machine (SVM) classifiers, three different variations were utilized in this implementation.

2.2.3.a SVM with Default Settings:

The SVM classifier was instantiated without specifying the kernel type, resulting in the default behavior of using the radial basis function (RBF) kernel. Cross-validation was performed using the cross_val_score function with 10-fold cross-validation. The resulting scores were stored in svm_scores, providing an evaluation of the default SVM model.

2.2.3.b SVM with Polynomial Kernel:

Another SVM classifier was created, this time with a polynomial kernel of degree 6. The degree parameter determines the degree of the polynomial kernel function. Cross-validation was again applied, and the resulting scores were stored in $\mathrm{svm}_{p}oly_{s}cores$.

2.2.3.c SVM with Sigmoid Kernel:

A third SVM classifier was implemented, utilizing the sigmoid kernel with a gamma value of 0.01 and a coef0 value of 0.5. The gamma and coef0 parameters control the influence of the kernel and the bias term, respectively. Cross-validation was performed, and the resulting scores were stored in svm_sigmoid_scores.

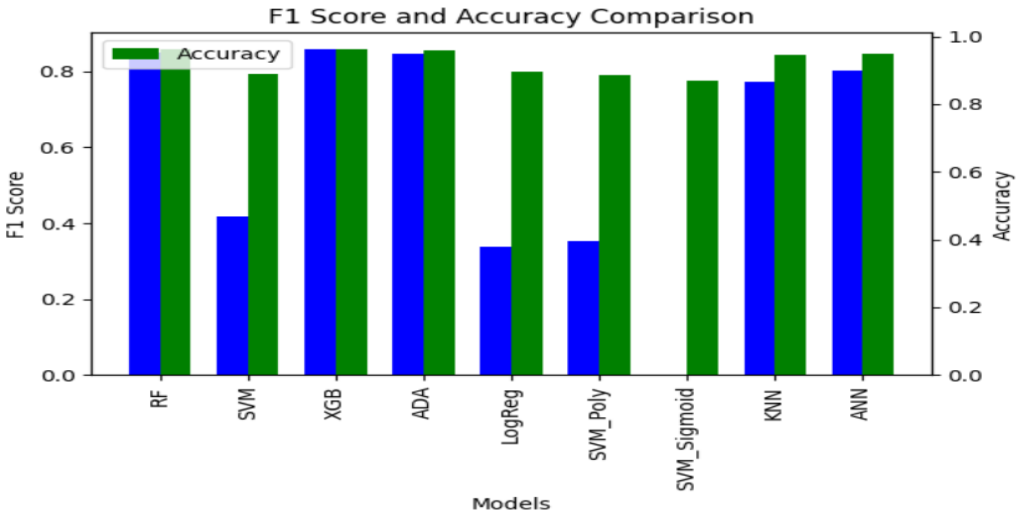
2.2.4 Other models

To evaluate the performance of different classification algorithms, namely XGBoost (XGBClassifier), AdaBoost (AdaBoostClassifier), and Logistic Regression (LogisticRegression), a cross-validation approach was implemented. Each classifier was trained and tested using the

cross_val_score function with 10-fold cross-validation. The resulting scores for each classifier were stored in xgb_scores, ada_scores, and logreg_scores, respectively. These scores provide an indication of the predictive accuracy of each algorithm and serve as a basis for comparing their performance in the binary classification task.

2.3 Performance comparison

The performance of multiple classifiers, including Random Forest, Support Vector Machine, XGBoost, AdaBoost, Logistic Regression, SVM with polynomial kernel, SVM with sigmoid kernel, K-Nearest Neighbors, and Artificial Neural Network, was evaluated using various metrics. For each classifier, the model was trained on the training data and used to predict the labels for the test data. The resulting predictions were compared to the true labels to calculate metrics such as the confusion matrix, F1 score, and accuracy. The F1 scores and accuracies were recorded for each classifier, providing a basis for performance comparison.



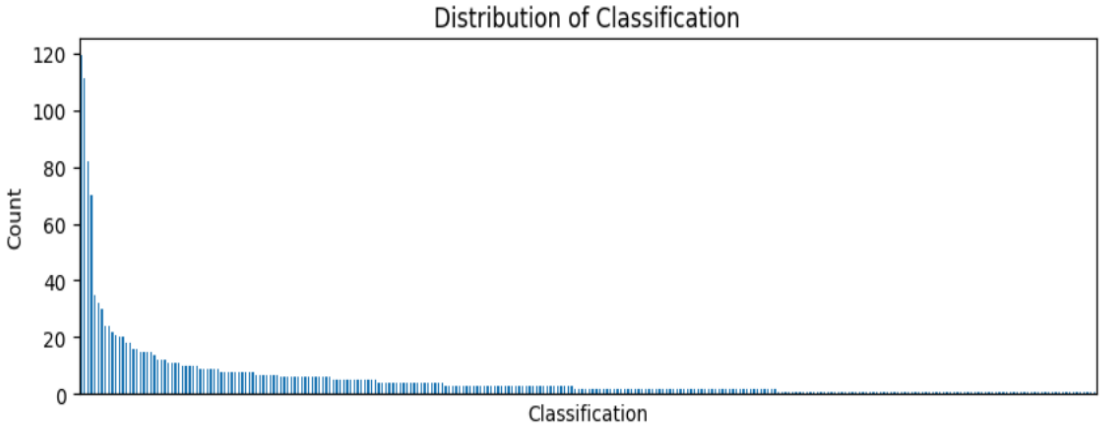
We can see that among the classifiers, Random Forest achieved a high F1 score of 0.85, indicating good predictive performance. XGBoost also performed well with an F1 score of 0.86. However, SVM with a polynomial kernel and sigmoid kernel had lower F1 scores of 0.42 and 0.0 respectively, suggesting poorer performance. Logistic Regression achieved a relatively low F1 score of 0.34. AdaBoost and the Artificial Neural Network showed decent performance with F1 scores of 0.85 and 0.80 respectively. K-Nearest Neighbors achieved an F1 score of 0.77.

These results highlight the varying performance of different classifiers in accurately classifying the binary target variable. However, due to the imbalanced nature of our dataset, relying on accuracy alone may be misleading. Instead, I evaluate the classifiers using the F1 score, which considers both precision (the ability to accurately identify positive instances) and recall (the ability to correctly identify all positive instances). The F1 score provides a balanced measure of performance, considering both the positive and negative instances, and helps me make a more informed decision when selecting the best classifier for the problem.

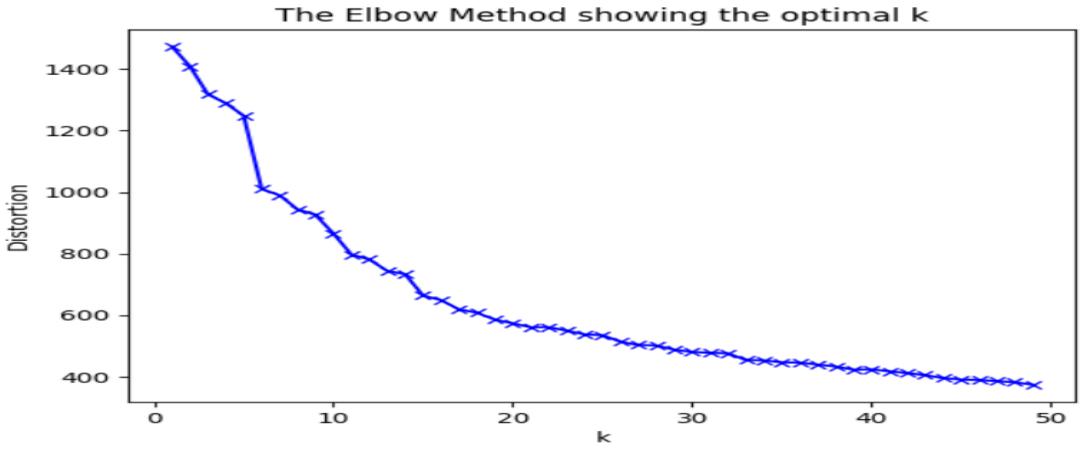
3 Clustering problem

3.1 Reduce number of classification classes

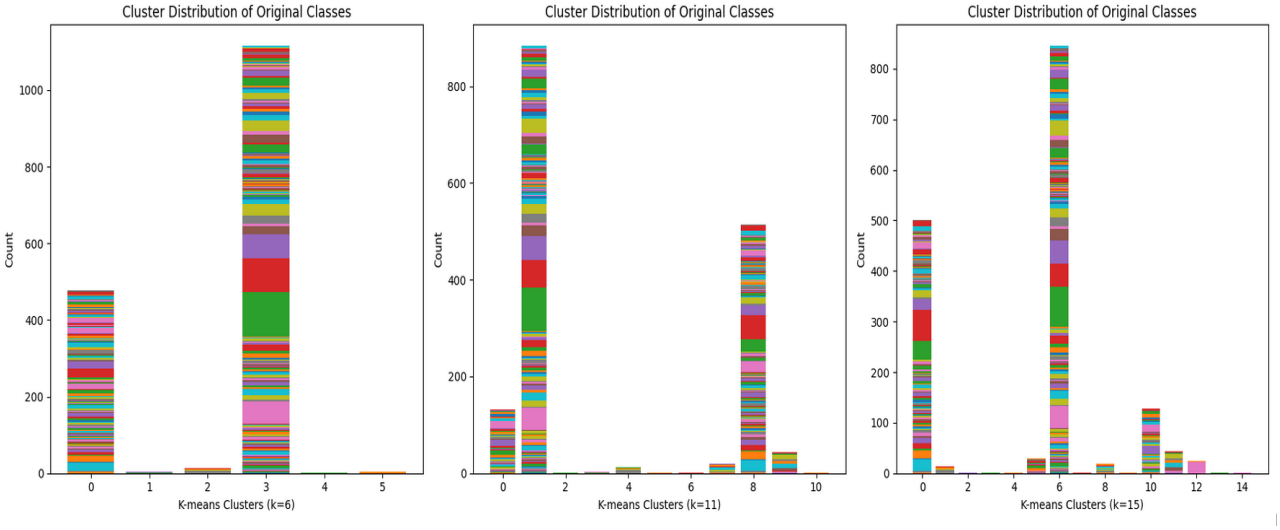
To address the imbalance in the original 'Classification' column, I employed dimensional reduction techniques to reduce the number of classification classes. The motivation behind this was to group similar classes together using the K-means algorithm with the Elbow method, aiming to minimize the error. After performing clustering, I reexamined the distribution using the original classes to assess the effectiveness of the grouping and its impact on the dataset.



To determine the optimal number of clusters (k) for the K-means algorithm, I calculated the distortion values for different values of k ranging from 1 to 50. For each k value, I initialized and fitted a K-means model using the KMeans function. The distortion, which represents the average Euclidean distance between data points and their assigned cluster centers, was then computed and stored in the distortions list. The result is:



Upon analyzing the distortion values, it is evident that there are three significant drops in distortion: from K = 1 to K = 6, from K = 11 to K = 11, and from K = 15. To further investigate these cases, I will plot the distributions for each of these three scenarios and examine the resulting clusters.



After examining the results, it appears that the distributions have improved slightly. However, there are still some classes with very few observations, which can be attributed to the initial class imbalance. Additionally, the presence of the curve of dimensionality could potentially impact the Euclidean distance calculation, leading to suboptimal results compared to expectations.

4 Dimensional reduction problem

4.1 Reduce number of features

To reduce the number of features, I utilized Principal Component Analysis (PCA) to condense the columns down to approximately the same number as the 6 core nutritional categories. By grouping related columns together and applying PCA, I aimed to capture the most important information while minimizing the dimensionality of the dataset. This approach helps in simplifying the feature space and potentially improving the performance of subsequent modeling tasks.

After experimenting with various techniques, such as considering cumulative variance ratio and absolute values, the resulting new features obtained through PCA did not appear to provide meaningful insights to me. Furthermore, when I applied the binary classification models mentioned earlier, there was no significant improvement in accuracy. This outcome suggests that the high dimensionality of the dataset and the loss of information during the dimensionality reduction process may have hindered the effectiveness of the models.

5 Conclusion

In summary, this project involved preprocessing steps, including handling null values and labeling data based on food form. Several models were evaluated, with Random Forest, XGBoost, and AdaBoost showing better performance. The impact of class imbalance was considered, leading to the use of F1 score as a more suitable metric than accuracy. Attempts to address class

imbalance and high dimensionality through clustering and dimensionality reduction did not yield significant improvements. Overall, the project emphasized the importance of preprocessing, model selection, and performance evaluation in binary classification tasks. Future work should explore more refined techniques for addressing class imbalance and high dimensionality.

6 References

References

[1] MACHINE LEARNING: A First Course for Engineers and Scientists - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön

[2] My code link:
https://colab.research.google.com/drive/1c2Af1ao_hijSVdA0OZuun8fRxBaK274x?usp=sharing