Fruit Quality Prediction Challenge

Dwaipayan Math
B. Tech (2023-2027), Heritage Institute of Technology, Kolkata
&
Tanzil Debbarma
BTMT (2022-2027), National Institute of Technology, Agartala

Project Guide: Prasun Dutta

Report submitted to: IDEAS – Institute of Data Engineering, Analytics and Science Foundation, ISI, Kolkata

1. Abstract

The "Fruit Quality Prediction Challenge" is a machine learning project aimed at developing a robust model to classify the quality of apples based on various attributes. This project uses advanced machine learning techniques to accurately predict apple quality, ensuring that only the best produce reaches consumers. By utilising a dataset containing features such as colour, size, weight, and taste attributes, the project aspires to create a comprehensive model that can distinguish between different quality grades. The primary objective is to enhance the efficiency and accuracy of quality control processes in the agricultural sector.

This initiative aims not only to improve quality assessment practices but also to reduce waste and increase profitability for apple producers. The findings from this project can be extended to other agricultural products, promoting the broader application of machine learning in agriculture.

2. Introduction

The agricultural industry, especially fruit production, significantly benefits from advancements in technology to ensure product quality and reduce waste. Quality assessment of apples is crucial as it affects market value, consumer satisfaction, and overall supply chain efficiency. Traditional methods of quality inspection are labour-intensive, time-consuming, and subjective. Implementing a machine learning-based classifier for apple quality prediction can provide a more consistent, efficient, and scalable solution, enhancing the overall quality control process in the agricultural sector.

Technology Involved:

1. **Machine Learning (ML)**:
   - Supervised learning algorithms for classification (e.g. Support Vector Machines(SVM), K-Nearest Neighbors(KNN), Multilayer Perceptron (MLP) and Gaussian Naïve Bayes(GNB)).
   - Feature selection and extraction techniques to identify key indicators of apple quality.
2. **Software Tools**:
   - Python and libraries such as scikit-learn training ML models.
   - Libraries like matplotlib to plot the graph of accuracy.

Procedure Used:

1. **Data Collection**:
   - Gather a dataset of apple images and sensor data representing various quality levels.
2. **Data Preprocessing**:
   - Split the data into training, validation, and test sets.
3. **Model Development**:
   - Experiment with various classification algorithms (e.g. SVM, KNeighborsClassifier, MLPClassifier, GaussianNB).
   - Perform hyperparameter tuning to optimize model performance.

4. **Model Evaluation**:
    - Conducting cross-validation to ensure the model's generalizability.

Purpose of Doing the Project:

The primary objective of this project is to develop an efficient and reliable machine learning model for predicting apple quality. The project aims to:

- Reduce labour costs and increase inspection throughout.
- Provide consistent and objective quality evaluations.
- Minimize post-harvest losses and improve supply chain efficiency.
- Enhance consumer satisfaction by ensuring high-quality produce.

This project not only addresses a critical challenge in the agricultural industry but also demonstrates the transformative potential of machine learning and AI in real-world applications.

3. Project Objective

The primary aim of the "Fruit Quality Prediction Challenge" project is to develop an efficient and reliable machine learning model for predicting the quality of apples. The objectives of this project are as follows:

1. **Develop an Accurate Classifier**:
    - Create a machine learning model capable of accurately classifying apples based on their quality metrics, such as colour, size, texture, taste and other physical attributes.
2. **Enhance Quality Control Processes**:
    - Automate the quality assessment process to reduce human error and subjectivity, ensuring a consistent and objective evaluation of apple quality.
3. **Optimize Supply Chain Efficiency**:
    - Improve supply chain operations by providing real-time quality predictions, thereby minimizing post-harvest losses and ensuring high-quality produce reaches the market.
4. **Reduce Labor Costs and Increase Throughput**:
    - Implement a scalable solution that can handle large volumes of apples, reducing the need for manual inspections and increasing the speed of the quality control process.
5. **Demonstrate the Application of Machine Learning in Agriculture**:
    - Illustrate the potential of machine learning and computer vision technologies in enhancing agricultural practices, specifically in the domain of fruit quality assessment and management.

By achieving these objectives, the project aims to showcase the transformative impact of advanced technologies in improving agricultural efficiency, product quality, and overall operational effectiveness.

4. Methodology

Procedure Used:

Data Collection:

A dataset from Kaggle is provided for using in our project.

Data Preprocessing:

Unnecessary columns and rows with missing data are removed using drop() function. The dataset is divided into train and test dataset having sizes ranging from 50% to 90%, using the train_test_split() function available in the sklearn library. The train and test dataset are further split into datasets containing features and targets of the apples.

Models Used:

The first algorithm used for training is SVM or Support Vector Machine.

"Support Vector Machine" (SVM) is a supervised learning machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems, such as text classification. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the optimal hyper-plane that differentiates the two classes very well (look at the below snapshot).

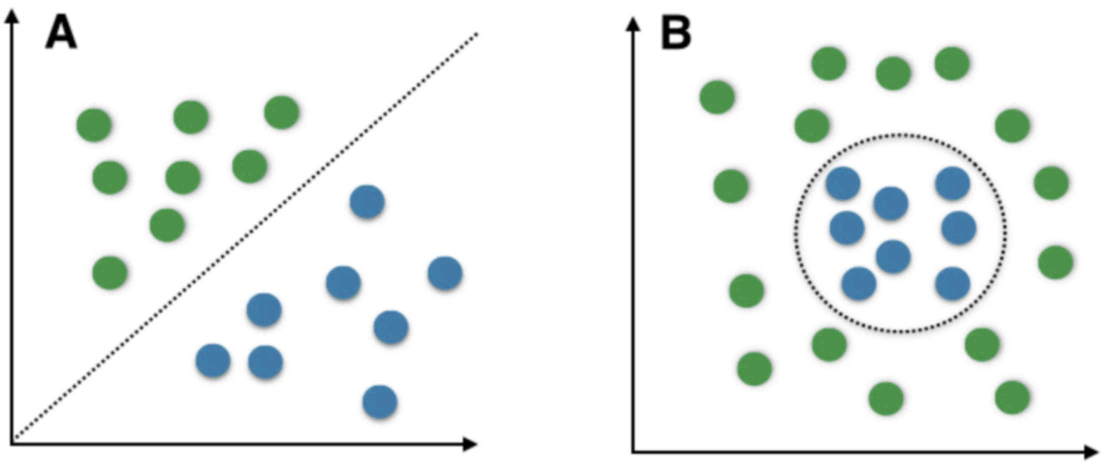

Fig: Hyper-plane differentiating two classes of data (green and blue)

Types of Support Vector Machines

Support vector machines are broadly classified into two types: simple or linear SVM and kernel or non-linear SVM.

1. Simple or linear SVM

A linear SVM refers to the SVM type used for classifying linearly separable data. This implies that when a dataset can be segregated into categories or classes with the help of a single straight line, it is termed a linear SVM, and the data is referred to as linearly distinct or separable. Moreover, the classifier that classifies such data is termed a linear SVM classifier.

A simple SVM is typically used to address classification and regression analysis problems.

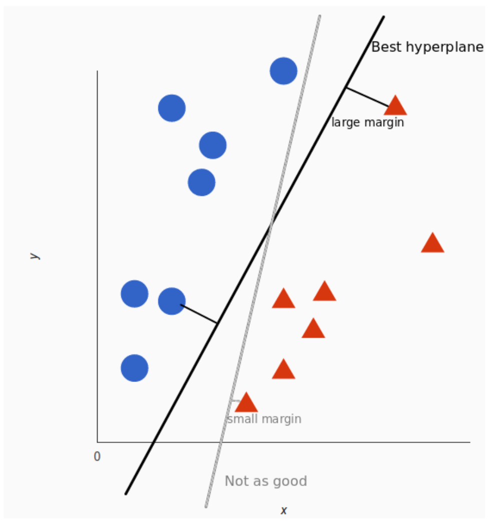

Fig: LinearSVM differentiating two classes of data (Red and Blue)

2. Kernel or non-linear SVM

Non-linear data that cannot be segregated into distinct categories with the help of a straight line is classified using a kernel or non-linear SVM. Here, the classifier is referred to as a non-linear classifier. The classification can be performed with a non-linear data type by adding features into higher dimensions rather than relying on 2D space. Here, the newly added features fit a hyperplane that helps easily separate classes or categories.

Kernel SVMs are typically used to handle optimization problems that have multiple variables.

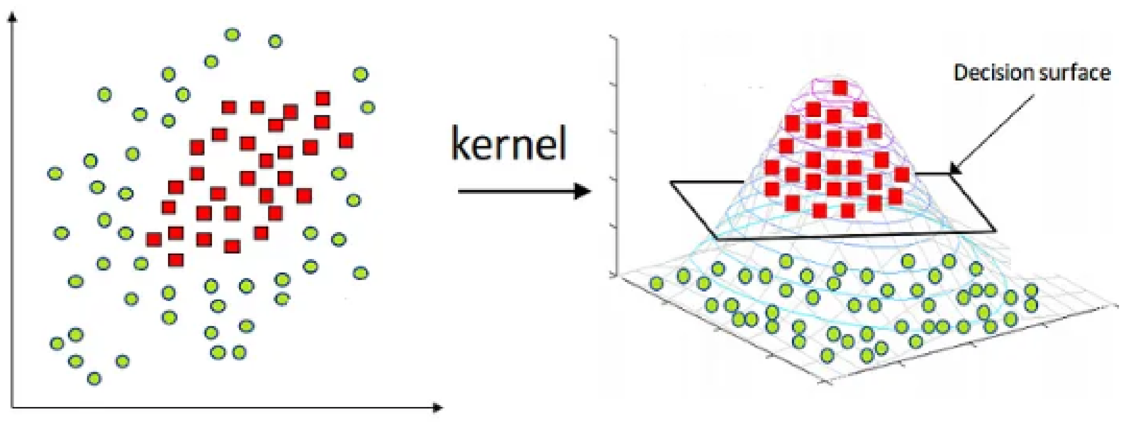

Fig: Kernel SVM 'rbf' creating a 3D hyperplane two differential Red and Green class of data

In our project, SVC is imported from "sklearn.svm". Then the train datasets are used to train the algorithm using different train data size everytime. Kernel 'rbf' or 'radial basis function' is used as it is giving the highest accuracy. The accuracy of the training is checked from the classification report.

<u>Hyperparameter tuning</u>:

Some parameters are used in the algorithm to increase its overall accuracy.

- C (Regularization parameter) = It controls the distance between decision boundary and closest data point. It also minimizes classification error of training data. In this project, C = 4.0
- Kernel = Type of pattern of the decision boundary. Kernel = 'rbf' or radial bound function is used in our project.
- Gamma = Helps determining the shape of the boundary. Gamma = 'auto' in our project.

The second algorithm is the K-Nearest Neighbor Classifier.

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm.

How does it work?

K is the number of nearest neighbours to use. For classification, a majority vote is used to determine which class a new observation should fall into. Larger values of K are often more robust to outliers and produce more stable decision boundaries than very small values (K=3 would be better than K=1, which might produce undesirable results.

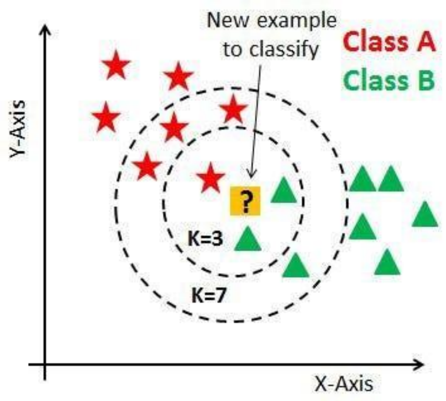

Fig: K-Nearest Neighbors where k = 3 and k = 7

In our project, KNN is imported from "sklearn.neighbors". The feature and target data of the train and test datasets is used to train the algorithm. The accuracy of the training is obtained in the classification report.

Hyperparameter tuning:

To increase the algorithm's efficiency, some parameters are used:

N_neighbors (value of k) = No. of neighbours to use for classification = 4

Weights: uniform (All points in the neighbourhood have equal weightage during classification).

Algorithm: auto

The third algorithm is MLP Classifier.

A multi-layer perceptron (MLP) is a class of feed forward artificial neural network. MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training.

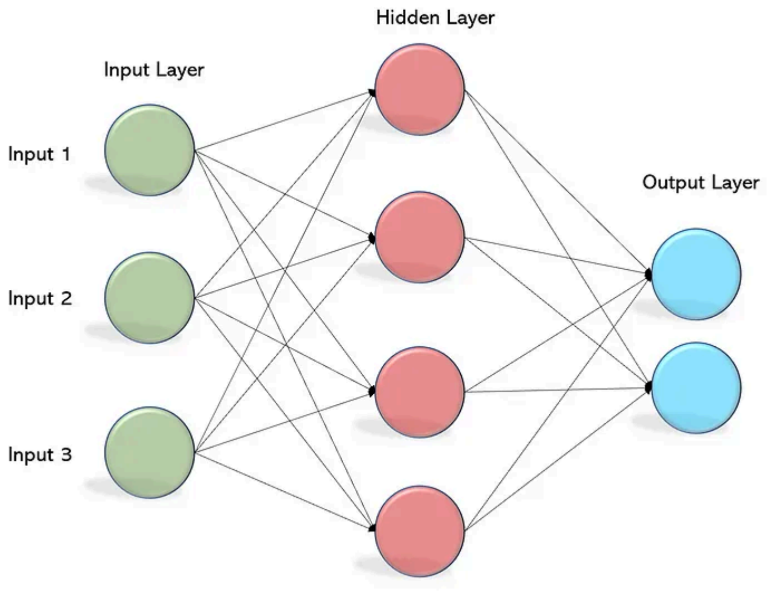

Fig: Multi-layer Perceptron

In our project, MLP Classifier is imported from "sklearn.neural_network". The feature and target data of the train and test datasets were used to train the algorithm. The accuracy of the training is obtained in the classification report.

<u>Parameters of MLP Classifier</u>:

- Solver ({'lbfgs', 'sgd', 'adam'}, default= 'adam'):

    The solver for weight optimization

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba
- Alpha (float, default=0.0001):

    Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss.

- Max_iter (int, default=200):

    Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

- Random_ state (int, default=None):

    Determines random number generation for weights and bias initialization, train-test split if early stopping is used, and batch sampling when solver = 'sgd' or 'adam'. Pass an int for reproducible results across multiple function calls.

<u>Hyperparameter tuning:</u>

To increase the algorithm's efficiency, some parameters are used:

Alpha = 0.01,

Max_iter = 300,

Random_state = 42

Finally, the fourth algorithm is Gaussian Naïve Bayes.

Gaussian Naïve Bayes is a machine learning classification technique based on a probabilistic approach that assumes each class follows a normal distribution. It assumes each parameter has an independent capacity of predicting the output variable. It is able to predict the probability of a dependent variable to be classified in each group.

The combination of the prediction for all parameters is the final prediction that returns a probability of the dependent variable to be classified in each group. The final classification is assigned to the group with the higher probability.

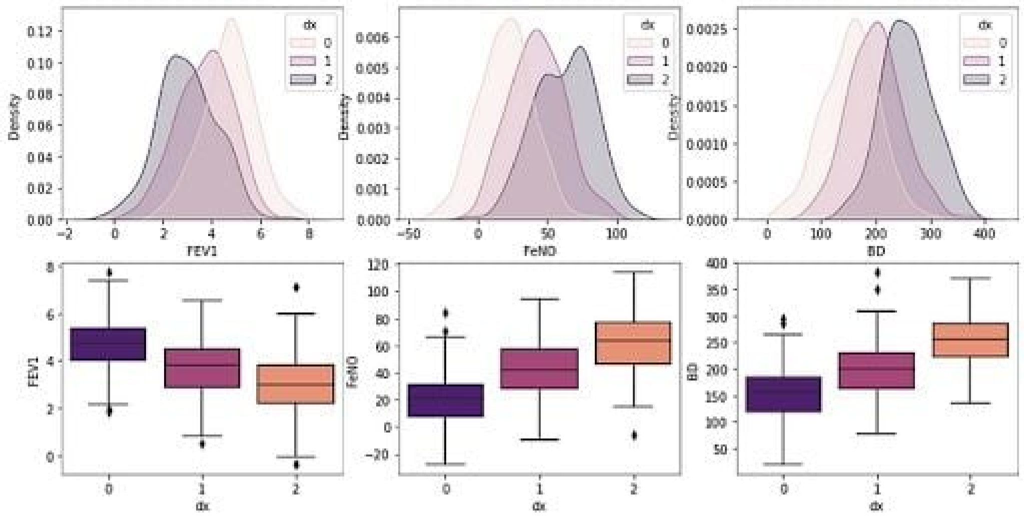

Fig: Gaussian Naïve Bayes

In our project, GaussianNB is imported from "sklearn.naive_bayes". The feature and target data of the train and test datasets are used to train the algorithm. The accuracy of the training is obtained in the classification report.

<u>Parameters of Gaussian Naïve Bayes</u>:

- Priors (array-like of shape (n_classes,), default=None):
  Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.
- var_smoothing (float, default=1e-9):
  Portion of the largest variance of all features that is added to variances for calculation stability.

<u>Hyperparameter tuning</u>:

To increase the algorithm's efficiency, some parameters are used:

Var_smoothing=1e-11

Model Evaluation:

Conducting k-fold cross-validation to ensure the model's generalizability.

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

In K-fold cross-validation, the data set is divided into a number of K-folds and used to assess the model's ability as new data become available. K represents the number of groups into which the data sample is divided.

Steps taken:

- **For using k-fold cross-validation for each classifier**

1) Importing Necessary Libraries

a) ‘SVC’, ‘KNeighborsClassifier’, ‘MLPClassifier’, ‘GaussianNB’: This is the classifier that will be used.

b) ‘train_test_split’: This function splits the dataset into training and testing sets.

c) ‘cross_val_score’: This function computes cross-validated scores.

d) ‘KFold’: This function provides a K-fold cross-validation iterator.

e) ‘numpy’: A package for scientific computing with Python.

2) Initializing the given Classifier

3) Preparing the Data

a) ‘X’: Feature matrix obtained by dropping the ‘Quality’ column from the dataframe

b) ‘y’: Target vector, which is the ‘Quality’ column.

4) Defining Training Sizes

a) ‘training_sizes’: Array of training set sizes, ranging from 40% to 80% of the dataset.

5) Loop through Training Sizes

For each ‘train_size’ in ‘training_sizes’:

1) Print the training set size as a percentage.

2) Split the data into training and testing sets using ‘train_test_split’ with the specified ‘train_size’ and a ‘random_state’ for reproducibility.

6) Define K-Fold Cross-Validation

A KFold object ‘kf’ is created for 5-fold cross-validation with shuffling and a fixed random state.

7) Perform Cross-Validation

a) ‘cross_val_score’ is used to perform K-fold cross-validation on the classification model using the training data ‘X_train’ and ‘Y_train’, with the KFold object ‘kf’.

b) The cross-validation scores are printed.

c) The mean cross-validation score is calculated and printed.

8) Cross-Validation Scores Output

The array contains the cross-validation scores for the given classification model for one of the training sizes. Each value represents the accuracy score for one of the k-fold splits.

● For plotting the results of k-fold cross-validation for different classifiers.

1) Import Libraries

a) ‘numpy’ is imported as ‘np’ for numerical operations.

b) ‘matplotlib.pyplot’ is imported as ‘plt’ for creating plots.

2) Create Training Sizes

a) ‘train_size’ is created using ‘np.arange’ which generates values from 0.4 to 0.8 (0.9 is not included) with a step of 0.1.

b) ‘x_values’ is obtained by multiplying ‘train_size’ by 100 to convert it into percentages.

3) Define Cross-Validation Scores and Labels

a) ‘y_values’ is a list containing the cross-validation scores for different classifiers. These variables should be defined elsewhere in the script.

b) ‘labels’ is a list of classifier names corresponding to the scores.

4) Set Figure Dimensions

Set the size of the plot to be 6 inches by 3 inches.

5) Plot the Data

This loop iterates through ‘y_values’ and ‘labels’ simultaneously, plotting each set of scores against ‘x_values’ and labelling the plot with the corresponding classifier name.

6) Set x and y Limits

Set the x-axis limits from 40 to 80 and the y-axis limits from 0.7 to 0.99.

7) Add Labels to Axes

Add labels to the x-axis and y-axis.

8) Add Legend

Add a legend to the plot with the specified number of columns (4) and location (‘upper center’).

9) Display the Plot

Use plt.show() to display the plot

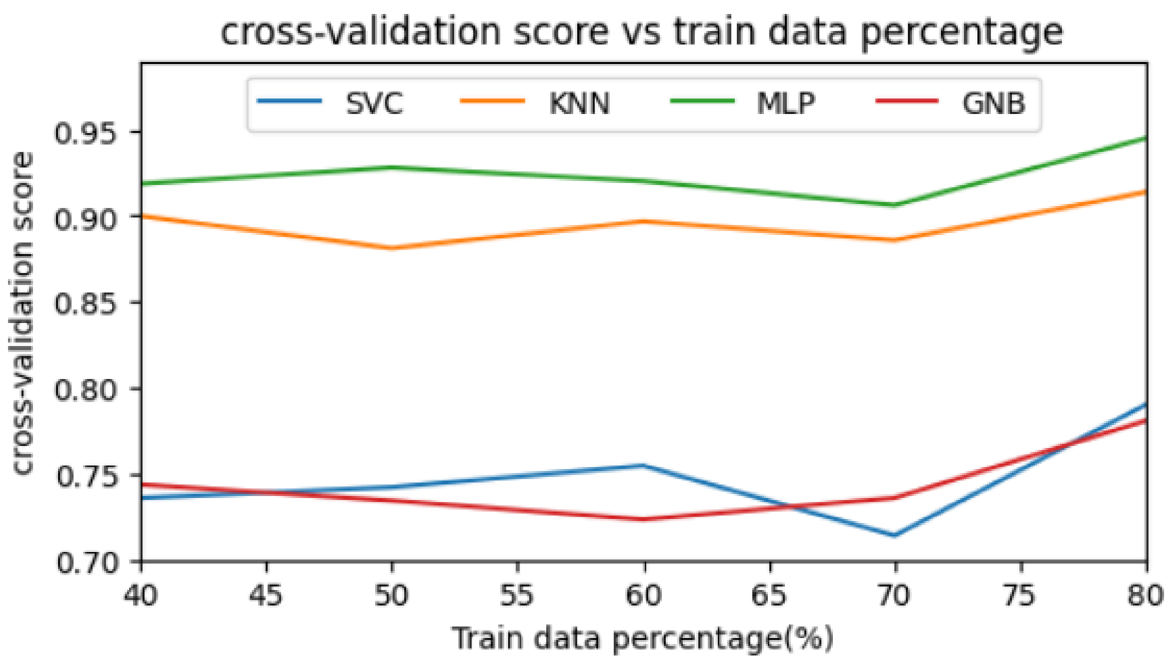

Fig: Cross-validation score vs Train data percentage graph

The above graph gives the performance of each of the four different classifiers as a function of the percentage of training data used. The x-axis represents the train data percentage ranging from 40% to 80%, while the y-axis represents the cross-validation score, which ranges from 0.70 to 0.95.

Here, the MLP classifier appears to have the highest and most stable cross-validation scores across different percentages of training data, while the GNB classifier shows the most variability. SVC shows improvement with training data, while KNN remains consistently high.

5. Data Analysis and Results

Classification_report() is used to evaluate the quality of training on both train and test data. The models' accuracy is plotted with respect to the size of test data used using matplotlib(), and the following curves are obtained:

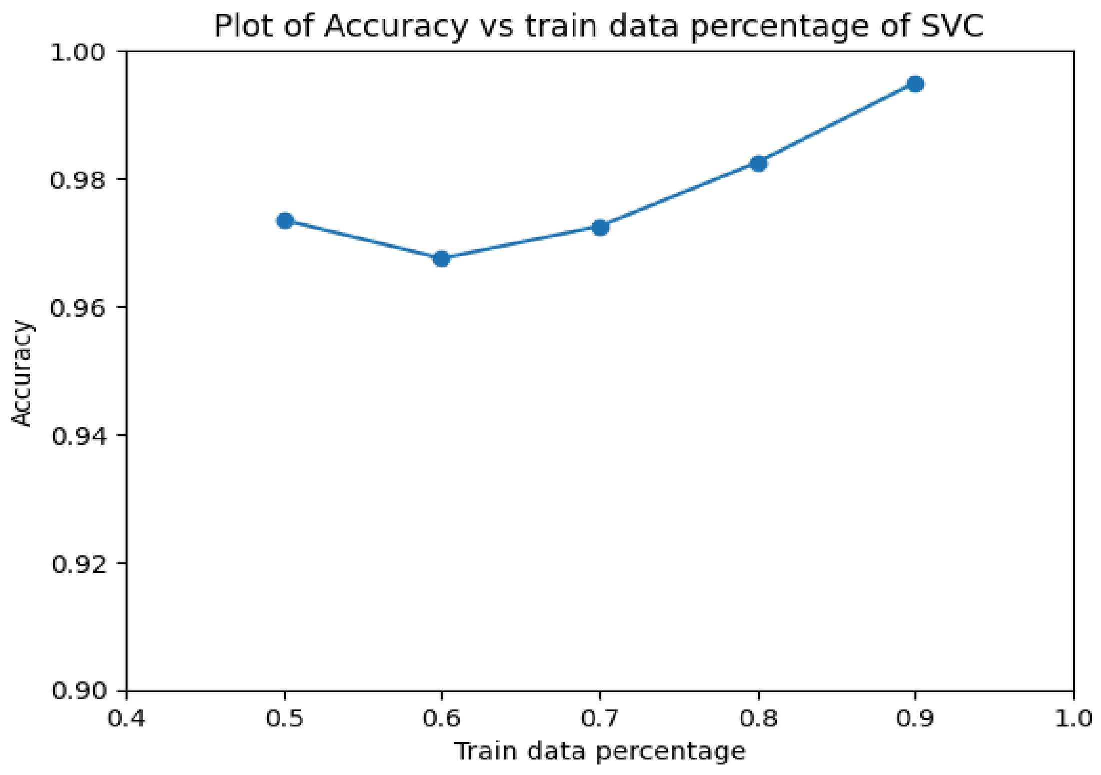

Fig: Plot of Accuracy vs train data size used in SVC

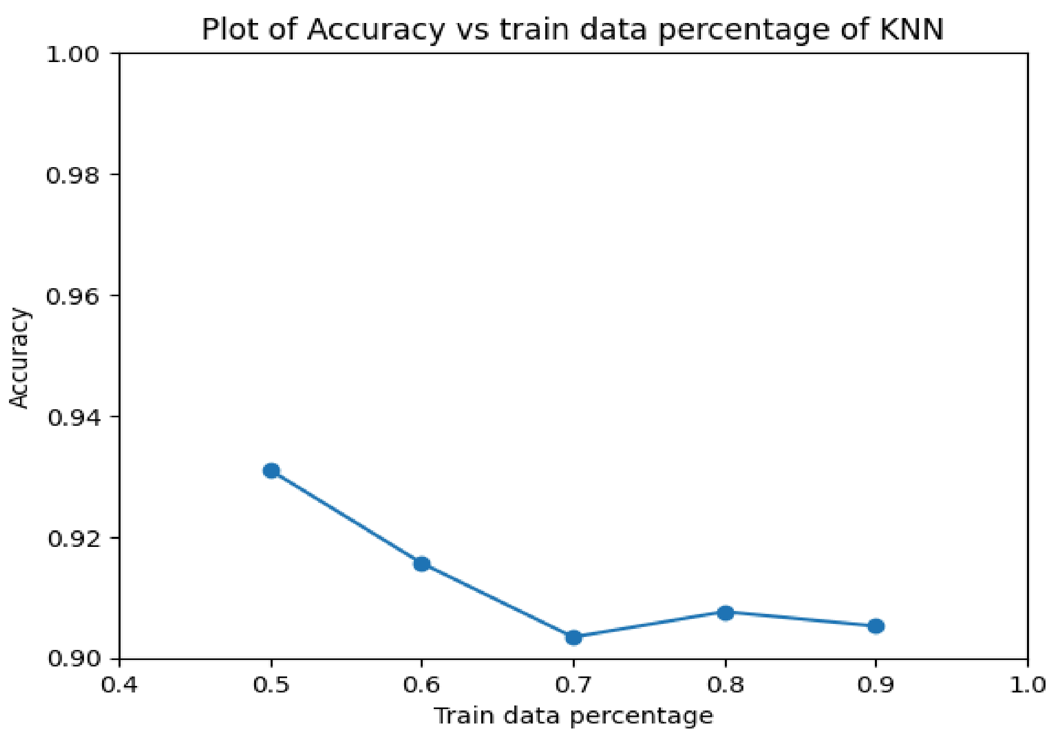

Fig: Plot of accuracy vs train data percentage in KNN

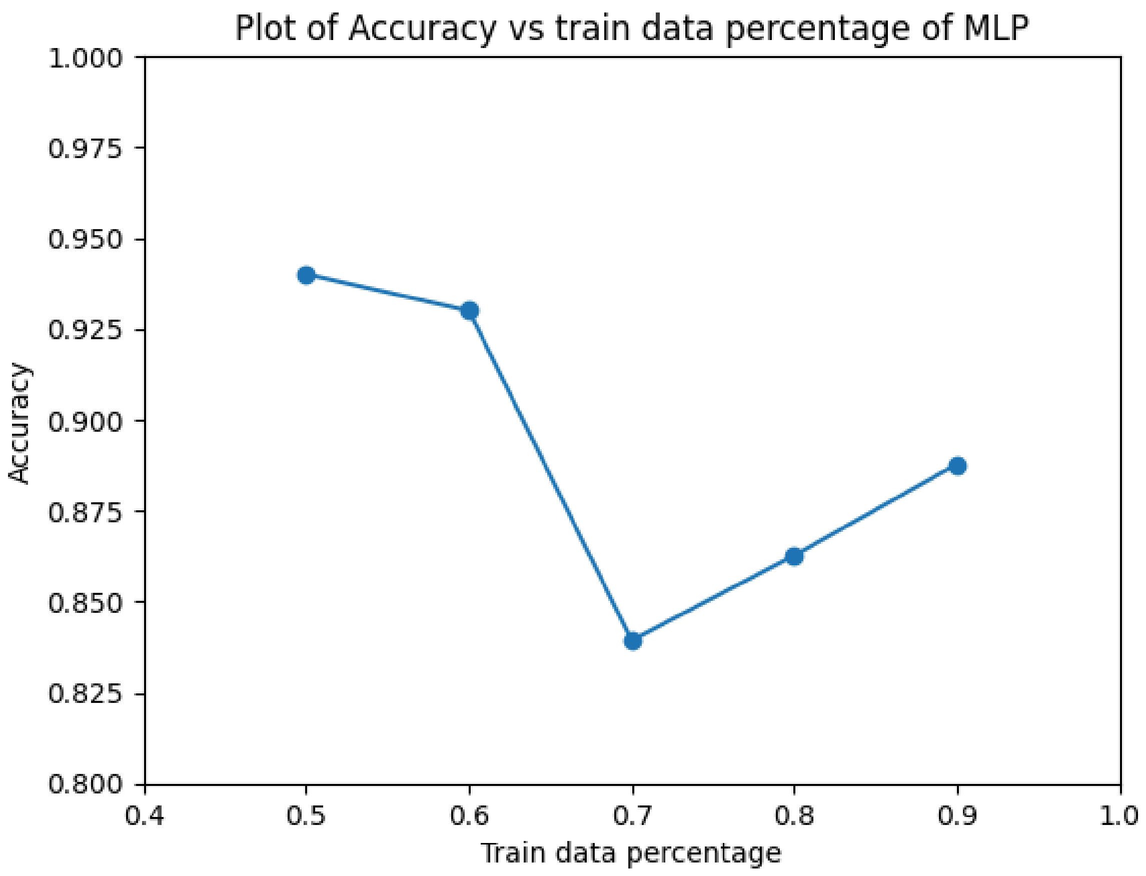

Fig: Plot of accuracy vs train data size in MLP

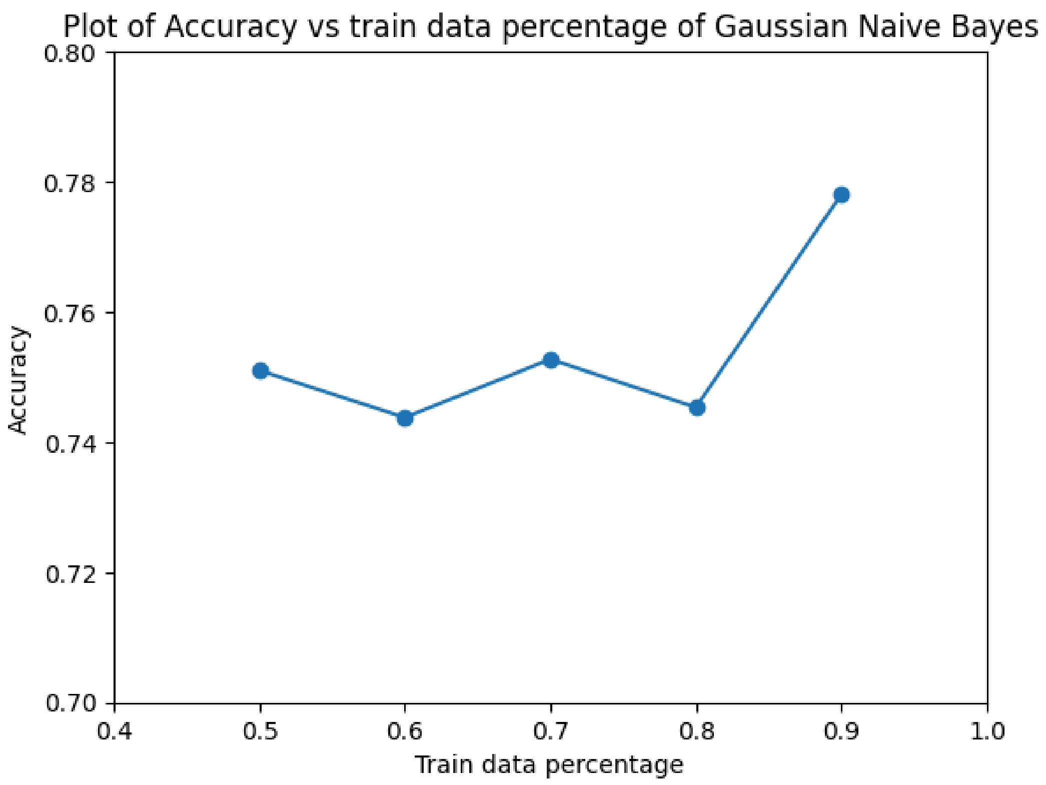

Fig: Plot of accuracy vs train data size in GNB

Thus, from the graphs, we can analyze the following:

- SVC gives the most accurate result, when 90% of the dataset is used to train the model.
- KNN gives the most accurate result, when 50% of the dataset is used to train the model, but its accuracy decreases gradually with increase in training data percentage.
- MLP also gives the most accurate result with 50% training data, but its accuracy drastically reduces at 70% training data size, and then increases gradually after that.
- For GNB, the changes are fluctuant, but the most accurate result is found with 90% training data size.

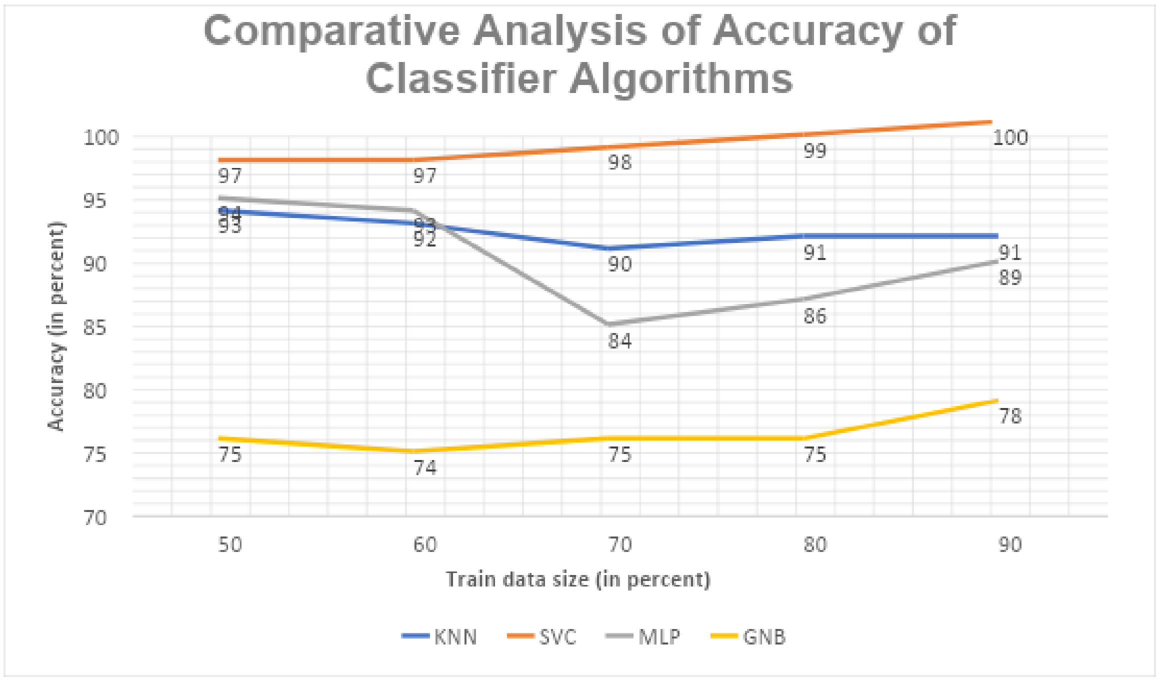

Fig: Line graph showing Comparative Analysis

Comparative analysis of accuracy of various classifier algorithms with varied training data size:

- At train data size = 50% & 60%, the accuracy from high to low:
  SVC > MLP > KNN > GNB.

- At train data size = 70%, 80%, 90%:
  SVC > KNN > MLP > GNB.

- Thus, on comparing all the algorithms, it is found that SVC gives the maximum accuracy for this type of dataset, and GNB gives the least accuracy.

6. Conclusion

Thus, to summarize our project, the following points can be mentioned:

- The "Fruit Quality Prediction Challenge" is a machine learning project aimed at developing a robust model to classify the quality of apples based on various attributes.

- Machine learning algorithms like SVC, KNN, MLP and GNB are involved. Python is used with libraries like pandas, numpy, sklearn and matplotlib.

- Primary objective of this project is to develop an efficient and reliable machine learning model for predicting the quality of apples.

- Procedure used included usage of dataset provided, Data preprocessing (splitting into train and test data, deleting unnecessary rows and columns), training ML algorithms with train data and checking their accuracy and precision with test data.

- Conducting k-fold cross-validation on the algorithms to check its score and plotting the graph of their performance.

- Plotting a graph using matplotlib, to visualize the accuracy of each model with various train data sizes, and analyzing which algorithm works best under a particular test data size.

7. APPENDICES

1. Code extract
   https://colab.research.google.com/drive/1vPtqq-6lyCiXRZApAIxIHWm6TipVb0KK#scrollTo=eqhFRNA4T2Un (Prediction using algorithms, classification report, plot of graph)
   https://colab.research.google.com/drive/1vjtTRTjMOrqHcTv7LL-Xg4YKWx3FFZVJ?usp=sharing (K-fold cross Validation, plot of graph)