CMPE 462 Term Project Report

Cem Güngör - 2020400219

Gülşen Sabak - 2020400072

Furkan Şenkal - 2020400249

1. WHY COFFEE BEANS?

- We all love coffee, and we were going to buy coffee while talking about our ideas. Then one of us said, “why don’t we select coffee as our topic.“ Hence, we chose coffee beans as our dataset.

2. DATA COLLECTION:

- The data collection procedure consists of different steps. First, we went through some websites and github repositories to have an idea about the attributes and how we can approach the problem. Websites include: https://varieties.worldcoffeeresearch.org/, https://www.kaggle.com/datasets/olesyaslonce/df-arabica-clean-2023/data, and many more.
- Then we decided on our attributes: Height, Width, Depth, Weight, Flavor, Acidity, Type, Process, Variety, Altitude, Origin, and Country... Then we ordered some coffee beans from the web, especially robusta beans since they are hard to come by and find. We mostly used arabica beans from our stock. Data is tabular and some attributes are taken from the features of the coffee whereas the others: weight, depth, height and width are measured by our hands. We used a scientific ruler for the length measurements. We have talked with the physics department and borrowed the scientific ruler. For weight measurements, we have used precision kitchen scales.


TOOLS


TOOLS


ON
OFF
START
STOP
0'00"
0.6
g

3. RAW DATA ON A TABLE

Feature	Number of Samples	Categories	Number of Samples in Each Category	Dimensionality
Height	1210	Continuous (numerical)	continuous data	1
Width	1210	Continuous (numerical)	continuous data	1
Depth	1210	Continuous (numerical)	continuous data	1
Weight	1210	Continuous (numerical)	continuous data	1
Flavor	1210	Categorical (e.g., "almond", "bitter")	almond: 130 bitter: 280 blackberry-nut-plum-vanilla: 130 chocolate-strawberry-grape-plum: 130 melon-pineapple-vanilla-grapes: 130 nectar-mango-forest fruit-grape: 130 woody-earthy: 280	1
Acidity	1210	Categorical (e.g., 1, 3)	1: 690 3: 260 4: 260	1
Type	1210	Categorical ("Arabica" and "Robusta")	Arabica: 650 Robusta: 560	1
Process	1210	Categorical (e.g., "natural", "Washed")	natural: 820 natural anaerobic: 260 washed: 130	1

Variety	1210	Categorical (e.g., "blend", "catuai")	blend: 130 bourbon: 260 catuai: 130 Catuai-IHECAFE90: 130 NARO-Kituza: 280 perdenia: 280	1
Altitude	1210	Categorical (e.g., "1700", "1200-1500")	1700: 390 1200-1500: 280 1800-2000: 130 500-1000: 280 750-1100: 130	1
Origin	1210	Categorical (e.g., "kasese", "Nord Kivu")	Chimaltenango: 130 Coorg/ Kodagu/ Chikmagalur-Western Ghats- Karnataka: 280 kasese: 280 Montecillos: 260 Nord Kivu: 130 ul de minas: 130	1
Country	1210	Categorical (e.g., "brazil", "India")	brazil: 130 Democratic Congo: 130 guatemala: 130 Honduras: 260 India: 280 uganda: 280	1

We had 7 different types of coffee beans, two of which belong to robusta and the remaining belong to arabica. Arabica beans have various varieties such as Bourbon, Catuai, or blend. Also, the tabular data has around 10 different attributes, and for each Arabica variety we measured and weighed 40 different beans whereas for each Robusta bean, we measured and weighed 100 different beans. After the data collection process was over, we decided to do data synthesis to create a more crowded and dense database and we increased the total number of coffee beans in our dataset to 1210. After data creation, for each arabica variety we had 130 different instances whereas for each robusta bean we had 280 different instances. While data synthesizing, we handled 7 different varieties separately and we put the 4 attributes we selected as a result of feature extraction into the Kolmogorov-Smirnov test. If the resulting p-value was greater than 0.05, we created new data with normal distribution. For attributes with a p-value less than 0.05, we synthesized new data using bootstrapping.

Here is the reason why we use this method:

We used the Kolmogorov-Smirnov (KS) test to check if the distributions of our selected attributes followed a normal distribution. The test compares the sample data to a normal distribution and gives a p-value. If the p-value is greater than 0.05, it means we can't reject the idea that the data follows a normal distribution. In this case, we generated synthetic data using a normal distribution, which is a reasonable choice because the original data seems to follow that pattern.

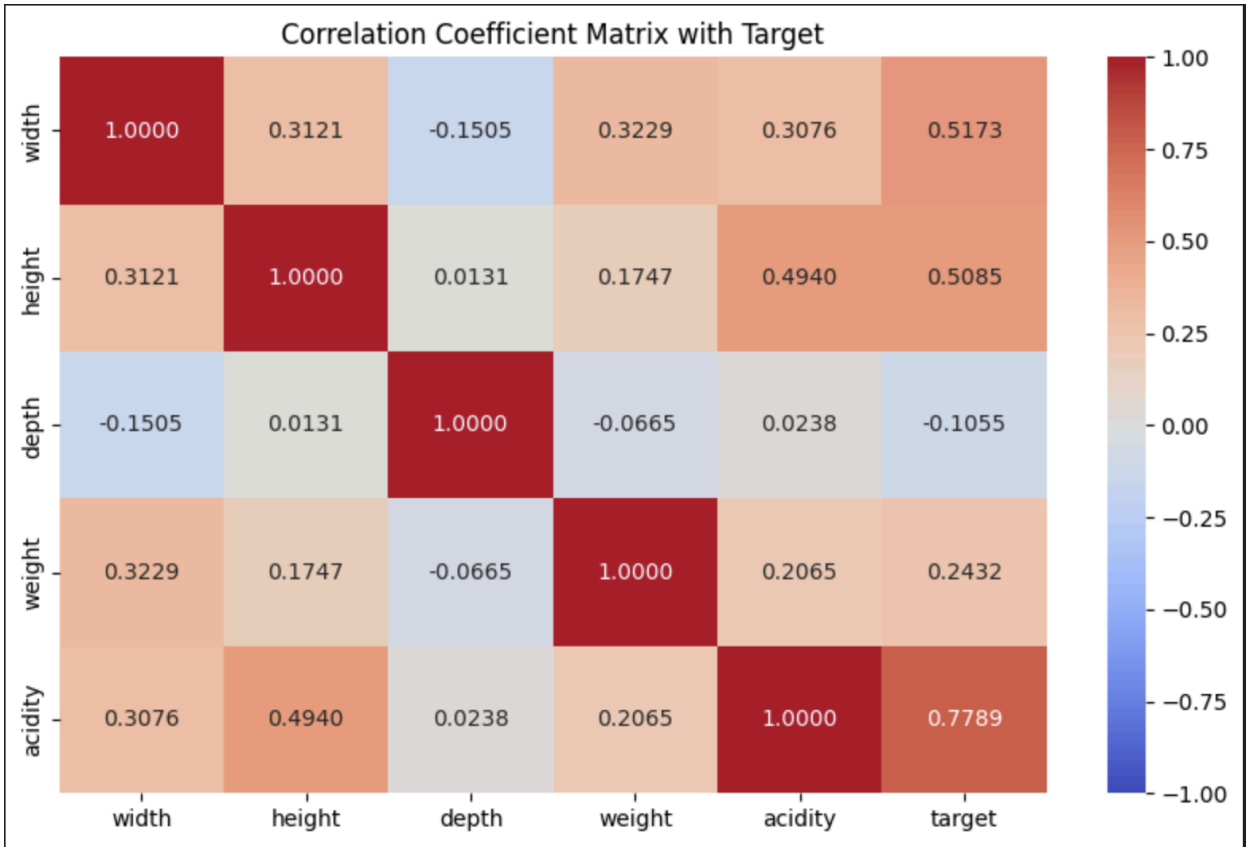
On the other hand, if the p-value was less than 0.05, it indicated that the data didn't follow a normal distribution. So, for those attributes, we used bootstrapping to generate new data. Bootstrapping works by resampling from the original data, which helps preserve the original distribution of the data, even if it's not normal.

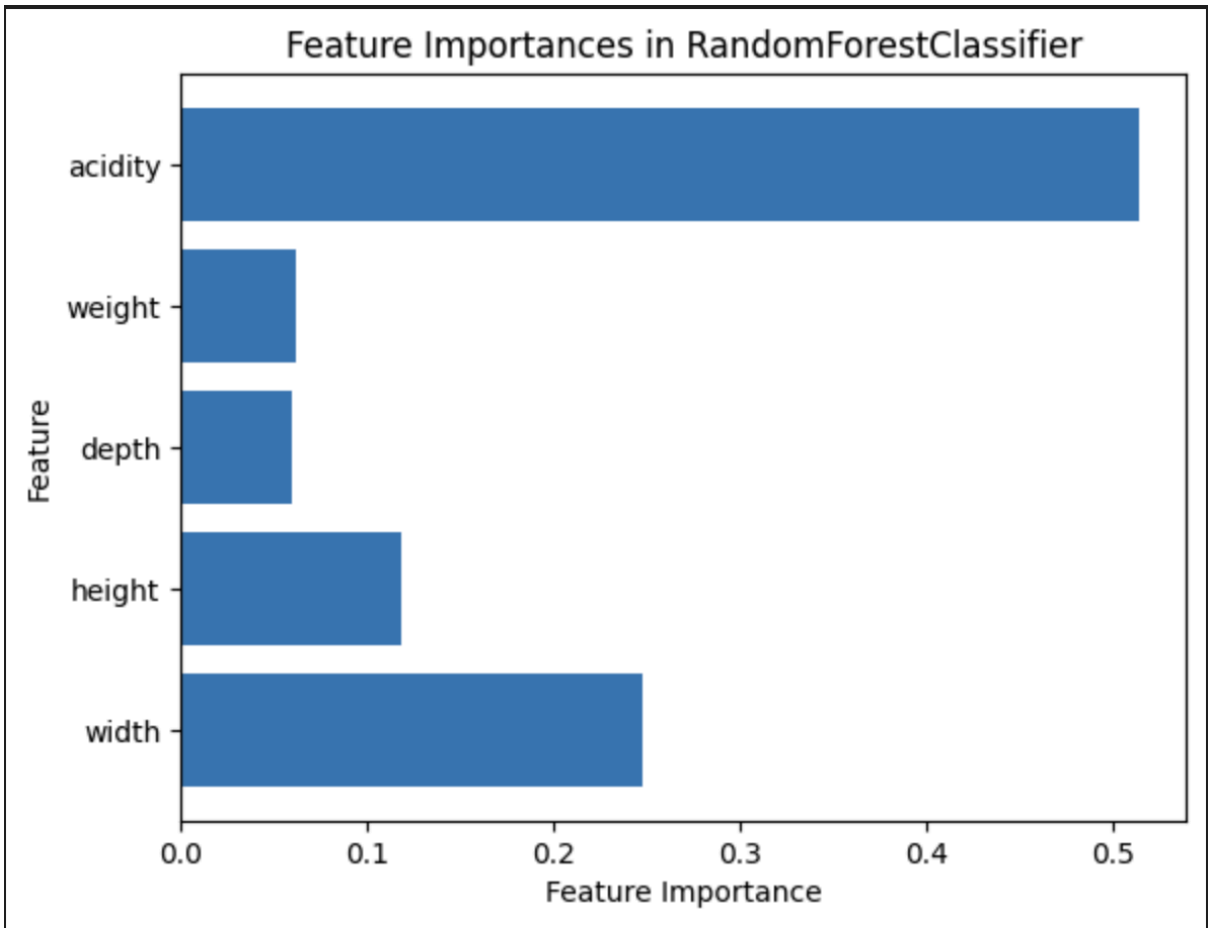
This approach helps us create new data that are consistent with the distribution of the original dataset, whether it follows a normal distribution or not, ensuring our synthetic data is as realistic as possible for further analysis.

Also, we used random.seed to ensure that the random functions we run (like generating synthetic data) produce the same results each time. Setting a seed makes the random number generator deterministic, meaning it will generate the same sequence of random numbers on every run. This is important for reproducibility so that if someone runs the same code with the same data and parameters, they get the same results. It helps in making the process consistent and ensures that the experiments can be replicated.

4. FEATURE EXTRACTION PROCEDURE

There are various approaches for classifying coffee beans. Most research approaches utilize image processing to classify coffee beans, but we wanted to do something different from what is available and decided to use tabular data to classify coffee beans based on their physical properties. We benefited from a few articles that you can find in the references section to find out what are the physical differences between arabica and robusta coffee beans. Our dataset contains 12 attributes. Five of the attributes are numerical, and the remaining are categorical. Numerical attributes are mainly physical properties of coffee beans, except for the acidity levels. We extracted features using two different approaches. For numerical ones, we used correlation coefficients as well as a random forest classifier feature importances. Since acidity is highly correlated with the class attribute (type), and its feature importance is around 50% we dropped acidity. For categorical features, we used pandas libraries cross-tabulation function to simply create a frequency table of the attributes with their corresponding classes. Since all categorical attributes were highly determinative, we did not use any of them.


Correlation Coefficient Matrix with Target
width
height
depth
weight
acidity
1.0000
0.3121
-0.1505
0.3229
0.3076
0.5173
0.3121
1.0000
0.0131
0.1747
0.4940
0.5085
-0.1505
0.0131
1.0000
-0.0665
0.0238
-0.1055
0.3229
0.1747
-0.0665
1.0000
0.2065
0.2432
0.3076
0.4940
0.0238
0.2065
1.0000
0.7789
width
height
depth
weight
acidity
target
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



type	arabica	robusta
altitude		
1700	390	0
1200-1500	0	280
1800-2000	130	0
500-1000	0	280
750 - 1100	130	0

type	arabica	robusta
variety		
Bourbon	130	0
Catuai, IHECAFE90	130	0
NARO-Kituza	0	280
blend	130	0
bourbon	130	0
catuai	130	0
perdenia	0	280

For categorical features, cross-tabulation illustrates the number of data points that belong to a class, given an attribute such as altitude, variety, etc. For example, the first picture includes altitude values and all data points between 1800-2000 are classified as arabica. This applies to other levels of altitudes, hence there is no use for altitude in classification for the given dataset.

type	arabica	robusta
process		
Natural Anaerobic	130	0
Natural anaerobic	130	0
Washed	130	0
natural	260	560

type	arabica	robusta
flavor		
Blackberry - nut - plum - vanilla	130	0
Chocolate - Strawberry - Grape - Plum	130	0
Nectar - mango - forest fruit - grape	130	0
almond - caramel - fresh tobacco	130	0
bitter - malt - hazelnut	0	280
melon - pineapple - vanilla - grapes	130	0
woody - earthy	0	280

type	arabica	robusta
country		
Democratic Congo	130	0
Honduras	260	0
India	0	280
brazil	130	0
guatemala	130	0
uganda	0	280

type	arabica	robusta
origin		
Chimaltenango	130	0
Coorg/ Kodagu/ Chikmagalur, Western Ghats, Karn...	0	280
Montecillos	260	0
Nord Kivu	130	0
kasese	0	280
sul de minas	130	0

5. DATA PREPROCESSING

Encoding Categorical Variables:

- For the **Acidity** feature, we manually (by hand) encoded the acidity levels into numerical values like 1, 3, and 4.
- The **Type** feature, which has categories like "Arabica" and "Robusta," was also encoded into numeric values in our code so that machine learning models could work with the data.

Scaling of Numerical Data:

- We applied scaling to the numerical features (**Height**, **Width**, **Depth**, and **Weight**) using standard techniques. This step is important to ensure that all the numerical features are on the same scale, which makes the model training process more efficient and helps improve the model's performance.

6. INTRA-CLASS AND INTER-CLASS SIMILARITIES

The intra- and inter-class similarities demonstrate how compact and distinct each class is from one another.
Intra-Class Similarity: This refers to the similarity or closeness between data points **within the same class**. It measures how similar the items in a single class are to each other. A higher intra-class similarity indicates that the data points within that class are more alike or tightly grouped.

Inter-Class Similarity: This refers to the similarity or closeness between data points **from different classes**. It measures how similar or dissimilar the items from one class are compared to those from another class. A lower inter-class similarity means the classes are more distinct from one another.

In this analysis, we compared the intra-class (within each class) and inter-class (between classes) similarities between two types of coffee (Robusta and Arabica) using three different similarity metrics: **Euclidean Distance**, **Manhattan Distance**, and **Cosine Similarity**. Below are the results for each of these metrics:

1. Euclidean Distance (L2 Norm)

The Euclidean distance is a commonly used metric to measure the straight-line distance between two points in a multi-dimensional space. Smaller values indicate higher similarity.

Result:

- Mean Intra-Class Similarity (Robusta): 2.5899776083485926
- Mean Intra-Class Similarity (Arabica): 2.247596220366747
- Mean Inter-Class Similarity (Robusta vs Arabica): 2.8411348242605876

Interpretation:

- The **intra-class similarity** for **Robusta** (2.59) is slightly higher than for **Arabica** (2.25), suggesting that Robusta's data points are closer to each other on average than Arabica's.
- The **inter-class similarity** (2.84) suggests that **Robusta** and **Arabica** are distinguishable in the feature space, but the distance between them is not very large. This indicates some overlap between the two classes.

2. Manhattan Distance (L1 Norm)

Manhattan distance, also known as L1 distance, calculates the absolute sum of the differences between corresponding features. It's useful when you care more about individual feature differences.

Result:

- Mean Intra-Class Similarity (Robusta): 4.172070757759082
- Mean Intra-Class Similarity (Arabica): 3.6861742640973105
- Mean Inter-Class Similarity (Robusta vs Arabica): 4.76755682989604

Interpretation:

- The **intra-class similarity** for **Robusta** (4.17) is again higher than for **Arabica** (3.69), indicating that the data points within each class are more spread out when considering the sum of feature differences.
- The **inter-class similarity** (4.77) indicates a notable difference between the two classes. It suggests that **Robusta** and **Arabica** are quite distinct from each other when considering the absolute differences in features.

3. Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors, with values closer to 1 indicating high similarity and values closer to -1 indicating dissimilarity. This metric is often used in text mining and high-dimensional data where the magnitude of the vectors might not be as important as their direction.

Result:

- Mean Intra-Class Similarity (Robusta): 0.10574167576080255
- Mean Intra-Class Similarity (Arabica): 0.16930787444266063
- Mean Inter-Class Similarity (Robusta vs Arabica): -0.12573756099922404

Interpretation:

- The intra-class similarity values for both Robusta (0.11) and Arabica (0.17) are quite low, indicating that the data points within each class are not very aligned in terms of direction, which is typical in multi-dimensional feature spaces.
- The inter-class similarity (-0.13) is negative, suggesting that Robusta and Arabica are not only dissimilar but also oriented in opposite directions in the feature space. This negative value further indicates that the two classes are distinctly different in terms of their feature vector directions.

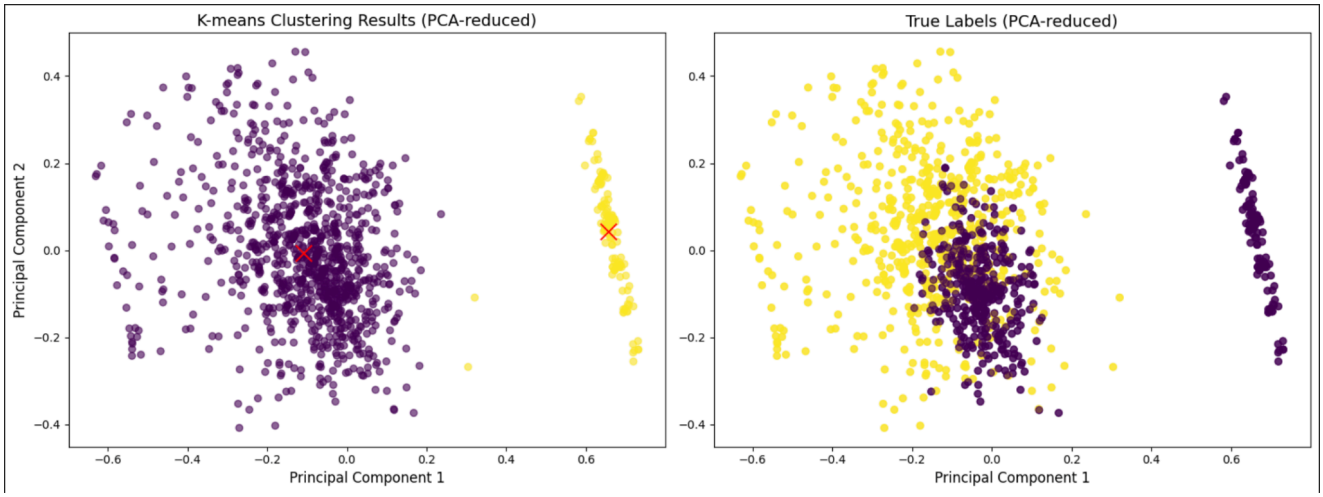
Overall Conclusion:

Inter-Class Comparison: The inter-class similarity values indicate that Robusta and Arabica are distinguishable, though the Euclidean distance suggests some overlap. Manhattan and Cosine distances show more distinct separation between the two classes, with Manhattan showing the highest separation (4.77), and Cosine similarity indicating that the classes are oriented in opposite directions (-0.13).

Main Conclusion: Although both classes are distinct, the degree of similarity or separation varies depending on the metric used. Robusta tends to be more cohesive within its class, while Arabica is slightly more spread out. The classes are most clearly separated using the **Manhattan and Cosine metrics**.

7. INTERPRETATION OF CLUSTERING (K MEANS)

The k-means clustering results with two clusters (representing Arabica and Robusta) show that the dataset is moderately distinguishable using the provided features (width, height, depth, and weight). While the two clusters are fairly separated in the PCA-reduced plot, with distinct centroids, the yellow cluster (most likely Arabica) exhibits significant scatter, indicating greater variability and less cohesion than the denser purple cluster(most likely robusta). This variability suggests that Arabica samples have more diverse physical attributes, which might create difficulties for classification. Although the dataset's features have reasonable discriminatory power, the overlap and variability in one class make it more difficult to classify perfectly, indicating some degree of complexity in distinguishing between the two categories.



8. RESULT OF MODELS

LOGISTIC REGRESSION

- **Logistic Regression (LR)** is a supervised learning algorithm used for binary classification tasks. It models the probability that a given input belongs to a particular class using a logistic function (sigmoid), which outputs values between 0 and 1. The model is trained by adjusting the weights to minimize the cost function, typically using gradient descent.

- Overfit exist or not:

    - The model that we have written does not suffer from overfitting because the test accuracy (0.6529) is not significantly worse than the training accuracy (0.6488). In fact, the test accuracy is slightly better, indicating that the model generalizes well to unseen data.

- Stopping Criteria:
    - In machine learning, **stopping criteria** determine when to end the training process of an algorithm, such as logistic regression. Training stops once we've reached a point where further training doesn't result in significant improvements.
    - There exists some different types of stopping criteria. We have used the cost function change threshold and maximum iteration limit.

    - **Cost Function Change Threshold:**
        - **Criteria:** Training stops when the change in the cost function (or loss function) between two consecutive iterations is less than 10^−6

$$|J(\theta^{(t)}) - J(\theta^{(t-1)})| < 10^{-6}$$

        - J(θ(t)) is the cost at iteration t

    - **Maximum Iteration Limit:**
        - Criteria: If the cost function change doesn't fall below 10^−6 after a set number of iterations (for example, 1,000 iterations), training will stop.

By stopping training when the cost function's change is very small, we avoid overfitting, as the model stops learning irrelevant details of the training data once it's reached an optimal point. If the training continues beyond a reasonable number of iterations without significant improvement (i.e., the cost change is tiny), it can also prevent underfitting by ensuring the algorithm doesn't stop too early.

- Training and Test Accuracies for Logistic Regression from scratch:
    - Train Accuracy: 0.6488
    - Test Accuracy: 0.6529
    - Runtime: 1.5139 seconds

- Training and Test Accuracies for Logistic Regression in Scikit-Learn:
    - Train Accuracy: 0.7634
    - Test Accuracy: 0.8388
    - Runtime: 0.0023 seconds
- Comparison of logistic regressions:
    - We have an algorithm that is worse than Scikit-learn's logistic regression function in terms of speed and accuracy, but it still completes in a tolerable time and has satisfactory accuracy.
    - The reason why from scratch is slower: Scikit-learn is highly optimized and implemented in C, which is much faster than a Python-based custom implementation. The from-scratch implementation, being written without such optimizations, is slower and takes more time to complete the training process.

SUPPORT VECTOR MACHINES

- A **Support Vector Machine (SVM)** is a supervised machine learning algorithm used for classification and regression tasks. It is particularly well-suited for problems with small to medium-sized datasets and works by finding the optimal boundary, called the **decision boundary**, that separates data points into different classes.

- Expression that we feed the solver
    - We solved:

$$\text{minimize} \quad \frac{1}{2}x^T Px + q^T x$$

$$\text{subject to} \quad Gx \leq h$$

$$\text{and optionally} \quad Ax = b$$

    - **1. P Matrix:**
        - Size: (d+1+N)×(d+1+N), where d is the number of features, and N is the number of training samples.

- Meaning: Defines the quadratic term of the objective function. Only the weights w are penalized, not the bias b or the slack variables ξ.

$$P = \begin{bmatrix} I_d & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Where Id is the identity matrix of size d×d.

- **q Vector:**
  - Size: (d+1+N)×1
  - Meaning: Defines the linear term of the objective function. Slack variables ξ are penalized with a factor C.

$$q = \begin{bmatrix} 0_d \\ 0 \\ C \cdot \mathbf{1}_N \end{bmatrix}$$

  - where 1N is a column vector of N ones

- **G Matrix:**
  - Size: (2N)×(d+1+N)
  - Meaning: Defines the inequality constraints:
    1. $y_i(w^T x_i + b) + \xi_i \geq 1$ (written as $-y_i(w^T x_i + b) - \xi_i \leq -1$).
    2. $\xi_i \geq 0$ (slack variables are non-negative).

$$G = \begin{bmatrix} -y_1 x_1^T & -y_1 & -1 & 0 & \cdots & 0 \\ -y_2 x_2^T & -y_2 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -y_N x_N^T & -y_N & 0 & 0 & \cdots & -1 \\ 0 & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 \end{bmatrix}$$

- The top half (N×(d+1+N)) handles the SVM constraints, and the bottom half ensures $\xi \geq 0$.

- **h Vector:**
  - Size: (2N)×1(2N) \times 1(2N)×1
  - Meaning: The right-hand side of the inequality constraints.

$$h = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- The first N elements correspond to −1 for $y_i(w^T x_i + b) + \xi_i \geq 1$, and the last N elements are 0 for $\xi \geq 0$.

- **A and b matrices:**
  - These are used for equality constraints, but none for soft-margin SVM

$$A = \text{None}, \quad b = \text{None}$$

- **RESULT OF CUSTOM SVM AND VARIOUS SVM'S**
  - Custom SVM: The SVM that we implemented
    - Result:
      - Test Accuracy: 0.8678
      - Train Accuracy: 0.7944
      - Training Time: 0.6914 seconds
      - Number of Support Vectors: 5
      - Weight vector (w): [1.13674498 1.05522057 -0.16442498 0.15522254]
      - Bias term (b): 0.013285141925986578

  - Linear SVM: A support vector machine that uses a linear decision boundary to separate classes in the feature space.
    - Result:
      - Linear SVM Train Accuracy: 0.7955
      - Linear SVM Test Accuracy: 0.8678
      - Time: 0.0080 seconds

  - RBF SVM: A support vector machine that uses a radial basis function kernel to map data into a higher-dimensional space for better class separation.
    - Result:
      - RBF SVM Train Accuracy: 0.8771
      - RBF SVM Test Accuracy: 0.8760
      - Time: 0.0086 seconds

  - Polynomial SVM (degree=3): A support vector machine that uses a polynomial kernel of degree 3 to capture non-linear relationships between features and class labels.
    - Result:
      - Polynomial SVM (degree=3) Train Accuracy: 0.7727
      - Polynomial SVM (degree=3) Test Accuracy: 0.7603
      - Time: 0.0075 seconds

- HYPERPARAMETER TUNING:

- We implemented 5-fold cross-validation for hyperparameter tuning to ensure robust model evaluation and optimal parameter selection. In this approach, the dataset is divided into five equal parts, where four parts are used for training and the remaining part for validation. This process is repeated five times, with each fold serving as the validation set exactly once, and the performance metrics are averaged to assess the model's generalizability. By evaluating the model across multiple splits, we mitigate the risk of overfitting to a specific subset of data and obtain a more reliable estimate of the hyperparameters' performance. This technique enables us to systematically explore hyperparameter combinations, such as the regularization parameter or kernel-specific settings, to identify the configuration that achieves the best balance between accuracy and model complexity.
- Result:
  - Best C: 1.0
  - Best Cross-Validation Train Accuracy: 0.7955
  - Best Cross-Validation Test Accuracy: 0.8678

- PERFORMANCE AND RUNTIME DIFFERENCES BETWEEN CUSTOM SVM AND SCIKIT-LEARN SVM
  - **1. Accuracy**
    - Both implementations achieve the same **test accuracy** of 0.8678. This indicates that both methods produce a similar decision boundary for the given dataset.
    - Train accuracy of the custom SVM (0.7944) is noticeably lower than the test accuracy. This could suggest that the custom implementation avoids overfitting to the training data, potentially due to differences in how regularization is handled compared to Scikit-learn. Scikit-learn's SVM is highly optimized and may incorporate internal mechanisms, such as better regularization tuning, to achieve more consistent performance across training and test datasets.
  - **2. Training Time**
    - The custom SVM takes **0.6914 seconds**, while Scikit-learn's SVM trains in **0.008 seconds**.
    - **Custom SVM** likely uses a general-purpose **quadratic programming (QP) solver**, which is computationally more expensive and not specifically optimized for SVMs.
    - Scikit-learn's implementation uses the **LIBLINEAR library**, which is highly optimized for linear SVMs. LIBLINEAR employs methods like

coordinate descent and linear kernel-specific optimizations, making it significantly faster, especially for larger datasets. (**LIBLINEAR** is an open-source library specifically designed for large-scale linear classification. It provides highly efficient implementations of linear Support Vector Machines (SVMs) and logistic regression, optimized for speed and scalability. LIBLINEAR is particularly well-suited for problems involving large datasets with a high number of features.)

- Scikit-learn benefits from efficient matrix operations and parallelization, whereas our custom implementation might not leverage such optimizations.

RANDOM FOREST

Random forest is an ensemble learning algorithm that trains n decision trees by randomly sampling subsets of attributes and datasets. If used for classification, it classifies a data point using the majority vote of the decision trees trained. It does not require an extra train-test split while training decision trees because it utilizes bootstrapping. We used 5 fold cross validation to select the number of estimators: decision trees that will be used in the training. Since n=500 has the highest accuracy, it is used for the training. The figure on the right summarizes the details of Random Forest. It likely overfits the data since training accuracy is significantly lower than test accuracy.

```
Cross-validation accuracies for each configuration:
n_estimators=10: Accuracy=0.8688
n_estimators=50: Accuracy=0.8719
n_estimators=100: Accuracy=0.8750
n_estimators=200: Accuracy=0.8730
n_estimators=500: Accuracy=0.8802

Optimal number of trees: 500

Training Accuracy: 1.0000
Test Accuracy: 0.9091

Test Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.88      0.91       120
           1       0.89      0.93      0.91       122

    accuracy                           0.91       242
   macro avg       0.91      0.91      0.91       242
weighted avg       0.91      0.91      0.91       242


Training time: 0.40 seconds
```

KNN

- KNN (K-Nearest Neighbors) is a simple, non-parametric machine learning algorithm used for classification and regression tasks. It makes predictions based on the similarity of data points, using the concept of "neighbors."
- After we have implemented KNN from scratch we get the result of best k as 7
    - Best k: 7
    - Cross-Validation Accuracies: ['0.824', '0.824', '0.847', '0.847', '0.849', '0.849', '0.854', '0.851', '0.851', '0.845', '0.839', '0.840', '0.845', '0.846', '0.845', '0.841', '0.847', '0.849', '0.852', '0.853']

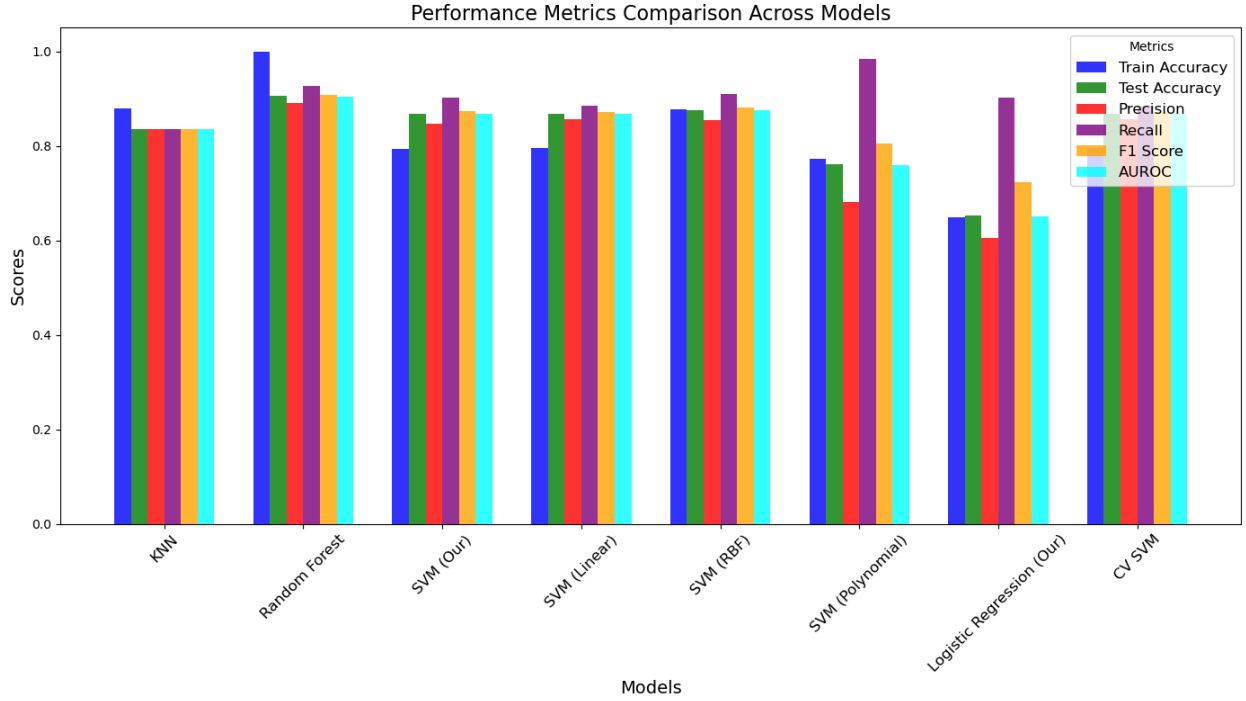
- Performance Metrics for KNN
    - Performance Metrics:
    - Train Accuracy: 0.8771
    - Test Accuracy: 0.8347
    - Precision: 0.8347
    - Recall: 0.8347
    - F1 Score: 0.8347
    - AUROC: 0.8347
    - Runtime (s): 0.5020

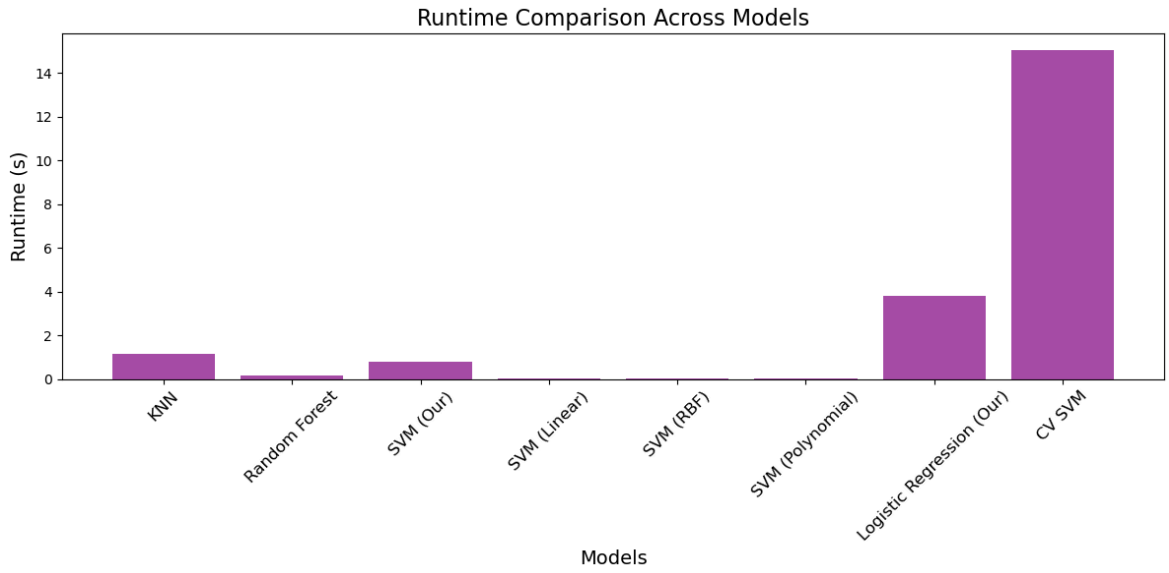
The final K-Nearest Neighbors model performed well on both the training and test sets, showing that it accurately captures the underlying patterns in the data while maintaining good generalization. The consistent cross-validation results demonstrate the model's stability across different subsets of the data, implying robustness. Furthermore, the model's precision, recall, F1 score, and AUROC are all balanced, indicating that it performs well in identifying both classes with minimal bias. Overall, the model is efficient, reliable, and appropriate for this dataset, reaching a good balance of computational cost and classification performance.

9. COMPARISON OF PERFORMANCES AND RUNTIMES OF EACH CLASSIFIER

Model	Test Accuracy	Train Accuracy	Training Time	Notes
Logistic Regression (from scratch)	0.6529	0.6488	3.8238 seconds	Limited performance with significant gap from Scikit-learn implementation
Logistic Regression (Scikit-learn)	0.8388	0.7634	0.0039 seconds	Efficient implementation with good balance of speed and accuracy
Custom SVM	0.8678	0.7944	0.7686 seconds	Strong accuracy but relatively slower training time
Linear SVM	0.8678	0.7955	0.0411 seconds	Matches Custom SVM accuracy with significantly faster training
RBF SVM	0.8760	0.8771	0.0475 seconds	Highest accuracy among SVM variants with minimal training time overhead
Polynomial SVM (degree=3)	0.7603	0.7727	0.0249 seconds	Underperforms compared to other SVM variants
SVM Cross Validation	0.8678	0.7955	0.7975 seconds	Provides robust validation with moderate computational cost
Random Forest	0.9050	1.0	0.1600 seconds	Perfect training accuracy indicates overfitting, but strong test performance
KNN (k=7)	0.8347	0.8791	1.1612 seconds	Moderate accuracy with longer inference time

	Model	Train Accuracy	Test Accuracy	Precision	Recall	F1 Score	AUROC	Runtime (s)
0	KNN	0.8791	0.834700	0.834700	0.834700	0.834700	0.834700	1.1612
1	Random Forest	1.0000	0.905000	0.889800	0.926200	0.907600	0.904800	0.1600
2	SVM (Our)	0.7944	0.867800	0.846154	0.901639	0.873016	0.867486	0.7686
3	SVM (Linear)	0.7955	0.867769	0.857143	0.885246	0.870968	0.867623	0.0411
4	SVM (RBF)	0.8771	0.876033	0.853846	0.909836	0.880952	0.875751	0.0475
5	SVM (Polynomial)	0.7727	0.760331	0.681818	0.983607	0.805369	0.758470	0.0249
6	Logistic Regression (Our)	0.6488	0.652900	0.604400	0.901600	0.723700	0.650800	3.8238
7	CV SVM	0.7955	0.867800	0.857143	0.885246	0.870968	0.867623	15.0336





10. EVALUATION METRICS

1. Classification Accuracy (ACC)

- **Definition**: Classification accuracy is the percentage of correct predictions out of all predictions made. It is calculated as:

$$\text{ACC} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

- **Focus**: Accuracy measures how often the model predicts the correct class, overall. It is a simple and intuitive metric.
- **Impact of Class Imbalance**: In the case of **class imbalance**, where one class is much more frequent than the other, accuracy can be misleading because the model can achieve high accuracy by simply predicting the majority class most of the time. For example, if 90% of the data belongs to one class, a model that predicts only that class will have an accuracy of 90%, but it won't be useful in correctly identifying the minority class.

2. Area Under the Receiver Operating Characteristic Curve (AUROC)

- **Definition**: AUROC is a performance metric that evaluates how well the model distinguishes between the classes. It is based on the **Receiver Operating Characteristic (ROC) curve**, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various thresholds.

$$\text{TPR (Sensitivity)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{FPR} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

- **Focus**: AUROC focuses on the model's ability to **rank** predictions correctly, regardless of the decision threshold. A higher AUROC indicates that the model is better at distinguishing between the positive and negative classes.

- **Impact of Class Imbalance**: AUROC is less sensitive to class imbalance and provides a balanced view of performance across various decision thresholds. However, it may still be misleading in cases of severe imbalance.

3. Average Precision

- **Definition of precision:** Precision is a metric that measures the proportion of **correct positive predictions** (True Positives) out of all the **predicted positive instances** (True Positives + False Positives). In other words, it tells you how many of the items that the model identified as positive are actually true positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Definition of Average Precision:** Average Precision is the mean of precision values calculated at different recall levels. It is often used in scenarios where you care about how well the model performs across different thresholds and is commonly used in the context of ranking or multi-class classification.

$$\text{Average Precision} = \frac{1}{N} \sum_{i=1}^{N} \text{Precision}(i)$$

(where N is the number of thresholds or recall levels, and precision(i) is the precision value at each recall level.)

- **Focus**: Average Precision focuses on how well the model maintains precision across different recall values. It is particularly useful when the model needs to prioritize identifying positives at different thresholds. This metric is sensitive to how the model ranks predictions.
- **Impact of Class Imbalance:** Average Precision is useful for imbalanced datasets as it focuses on the minority class, balancing precision and recall, making it a more realistic metric when identifying the minority class is critical.

4. Average Recall

- **Definition of recall:** Recall is a metric that measures the proportion of **actual positive instances** that were correctly identified by the model. In other words, it answers the question: "Of all the actual positives, how many did the model successfully identify?"

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **Definition of Average Recall: Average Recall** is a metric used to summarize the recall performance of a model across different thresholds. Recall, or True Positive Rate, measures the proportion of actual positive instances that the model correctly identifies. Average Recall calculates the mean of recall values at different thresholds or decision levels.

$$\text{Average Recall} = \frac{1}{N} \sum_{i=1}^{N} \text{Recall}(i)$$

(where N is the number of thresholds, and recall(i) is the recall value at each threshold.)

- **Focus:** Average Recall focuses on the model's ability to correctly identify all true positive instances, emphasizing the reduction of false negatives. It evaluates how well the model performs in detecting positive instances, particularly when decision thresholds are adjusted.
- **Impact of Class Imbalance:** In imbalanced datasets, Average Recall is more informative than accuracy because it emphasizes how well the model identifies the minority class.

5. F1- Score

- **Definition:** The F1-score is the harmonic mean of precision and recall, balancing the two metrics.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Focus:** The F1-score is particularly useful when the class distribution is **imbalanced** because it takes both false positives and false negatives into account, ensuring that the model performs well in both precision and recall. The higher the F1-score, the better the model is at handling both classes.

- **Limitations:** F1-score is a single number summary of precision and recall, so while it gives a more balanced view, it may not fully capture model performance in certain cases (e.g., very low or very high precision or recall).

11. BEST MODEL

- The **Random Forest** model performed the best overall, with the highest **test accuracy** (0.9050) and **train accuracy** (1.0). While this shows great performance on the test data, the perfect training accuracy suggests the model might be **overfitting**, meaning it memorized the training data too well. This is common in models that perform very well on the training data but may struggle when introduced to new, unseen data

- Why Random Forest Performed Well:
    - **Ensemble Method**: Random Forest is an ensemble learning technique, meaning it combines predictions from multiple decision trees to make a final prediction. This reduces the likelihood of overfitting that can happen with individual decision trees, and it makes the model more stable and accurate overall.
    - **Generalization**: Because it uses multiple trees, Random Forest is better at generalizing from training data compared to single decision trees, which might be too specific to the training set.
    - **Versatility**: Random Forest works well with many types of data and can capture complex patterns in data, making it useful for classification tasks like this one.
    - **Resistant to Noise**: The model is relatively immune to noise and outliers, which helps it perform better in real-world scenarios where data isn't always clean or perfectly structured.

- Why it might overfit:
    - Even though **Random Forest** has the best performance, its **perfect training accuracy** suggests it might be overfitting. This could happen if:
        - The model uses too many trees without tuning, making it too specific to the training data.
        - There's no regularization, like limiting the depth of trees or pruning them, which can result in overly complex models that perform well on the training set but fail to generalize to new data.

- Conclusion:
    - While **Random Forest** is the top performer in this case, the overfitting issue should be addressed, possibly by tuning the model's parameters (e.g., adjusting the number of trees or limiting tree depth). Other models like **RBF SVM** also performed well but may have trade-offs in terms of training time and their ability to generalize.

- Insights for Best Model:

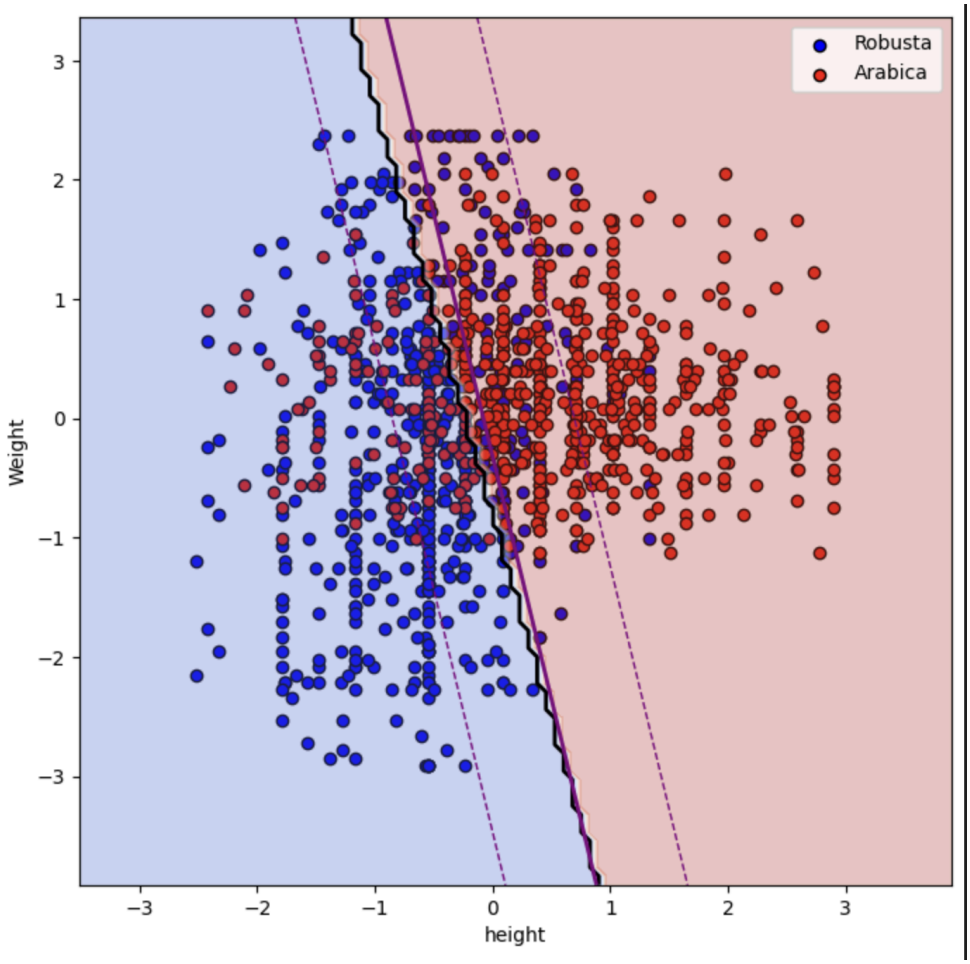
- Best Model for Performance: The Random Forest achieves the highest test accuracy (0.9050) but shows signs of overfitting with perfect training accuracy.
- Best Model for Speed: Scikit-learn's Logistic Regression (0.0039s) and Linear SVM (0.0411s) remain the fastest options.
- Best Model for Generalization: The RBF SVM shows excellent balance between training (0.8771) and test (0.8760) accuracy, suggesting good generalization.
- Computational Efficiency: SVM variants generally offer good performance-speed tradeoffs, with RBF SVM providing the best balance.

12. CHALLENGES

The first difficulty we encountered was the selection of the topic. Since we had to come up with our own topic and find data either by scraping or collecting we thought a lot on this part. Then the most challenging part of this assignment was collecting our own dataset and making it usable for the models. Measuring data, transferring to Excel as a tabular format, and cleaning took around 30% of the total time we had dedicated to the project. Also, another main challenge was synthesizing more data. We had to be very careful and think a lot while synthesizing the data and extracting the features. After fully preparing the dataset, one of the main challenges in the coding part was understanding the needs of the QP solver and preparing the data for the algorithm. Last but not least was writing logistic regression from scratch. These were the main challenges we faced while doing the project.

13. DECISION BOUNDARY:

We have selected two continuous attributes for plotting decision boundaries, weight and height. Although there are no visible outliers in the dataset, it can be seen from the plot that the dataset is not linearly separable. Regarding physical properties, coffee beans have a high variance, therefore some overlap exists. We do not expect linear models to perform well on the dataset. The purple line is the decision boundary of soft margin SVM, where dotted lines are +- 1 to the margin, whereas the black line shows the decision boundary of logistic regression. Exceeding our expectations, none of the models performed badly. SVM had a 77% accuracy score, outperforming logistic regression by 1%.



CONTRIBUTIONS

Cem Güngör - 2020400219

Data Collection: Weighed and measured natural-eljardin(40) and robusta kasese(100) beans. Also filled Excel with those beans. Found categorical attributes for each bean and updated the dataset with them. Researched about data synthesizing more data. Worked with Furkan for the feature extraction part to explain why categorical attributes are not used. Refactored the synthesizing part to constantly produce the same data and combined all synthesized and measured data to be used in algorithms. Visualized data points on k-means using both their true class and the class given by k-means. Worked on soft margin support vector machines to prepare the data with the team. Wrote the code to use cvxopt's solver. Worked on logistic regression with Gülşen using lecture notes. Tested whether the algorithm requires a regularization or not. Together with Furkan, compared logistic regression to SVM and created a meshgrid and plotted

the decision boundaries of the algorithms. Created comparison charts used throughout the report. Apart from the charts and tables, the report is split evenly between all team members.

Gülşen Sabak - 2020400072

Data Collection: Weighed and measured nord-kivu(40), guatemala-anaerobic(40), and honduras-anaerobic(40) beans. Also filled Excel with those beans. Coded the data synthesizing part to reproduce more data using the underlying distributions. Wrote the code for inter/intra-class similarities and explained how the dataset and the attributes used are not very similar to each other. Ran PCA on data points to be able to visualize data points on a 2D plot for k-means clustering tasks. Coded k-NN and k fold cross validation for k-NN to select the best k. Ran the algorithm ranging from 1 to 21 and used the best value to train k-NN. Worked on soft margin support vector machines to prepare the data with the team. Validated soft margin SVM to find the best value for C. Coded most of the logistic regression algorithm from scratch. Worked with Cem on some parts. Created the tables that are used throughout the report. Apart from the charts and tables, the report is split evenly between all team members.

Furkan Şenkal - 2020400249

Data Collection: Weighed and measured brazil-pinkstar(40), and indian-robusta(100) beans. Also filled Excel with those beans. Worked on the feature extraction part to create and visualize the correlation coefficient matrix and explained it. Coded k-means clustering from scratch and measured its accuracy among all data points. Wrote the code for the random forest classifier and used gridSearch to select the best hyperparameter for the number of decision trees. Evaluated the model based on the performance and run-time. Worked on soft margin support vector machines to prepare the data with the team. Compared SVM written by the team to scikit-learn's SVM using different kernel functions and calculated the metrics. Compared logistic regression written by the team with scikit-learn's logistic regression and calculated the metrics. Solved the problem of changing accuracy. Together with Cem, compared logistic regression to SVM. Created most of the bar charts that are used throughout the report. Apart from the charts and tables, the report is split evenly between all team members.

REFERENCES

- https://www.sciencedirect.com/science/article/abs/pii/S0021863499904110
- https://iopscience.iop.org/article/10.1088/1755-1315/1246/1/012056/pdf
- https://www.researchgate.net/publication/366044857_Comparative_study_of_changes_in_physical_mechanical_and_colour_properties_between_arabica_and_robusta_coffee_beans_prior_to_and_after_roasting

LINK TO ZIP:
https://drive.google.com/file/d/1-WZ0nOnt761pr8NToDBOx_6djfwd1nf9/view?usp=sharing