Predicting U.S. Legal Tender Coins using Machine Learning

Date: 03/16/2025

Author: Garrett Engelder

Table of Contents

Table of Contents Cont.

Table of Contents Cont.

Table of Contents Cont.

Table of Contents Cont.

Table of Contents Cont.

Table of Contents Cont.

Table of Contents Cont.

Abstract

This project applied machine learning techniques to predict whether a coin is legal U.S. tender or not based on its physical properties. The dataset consisted of 360 coins, evenly split between legal U.S. tender (180) and foreign tender (180). Key features included diameter, thickness, weight, edge type, magnetism, and main metal composition.

Preprocessing steps were carried out to prepare the data, including scaling numerical features, encoding categorical variables, and transforming attributes into dummy variables. Five machine learning models were trained and evaluated: K-Nearest Neighbors (KNN), Decision Trees (DT), Random Forests (RF), Perceptrons, and Logistic Regression (LR). Each model was tested with and without bagging classifiers to evaluate their impact on prediction accuracy. Hyperparameter tuning was performed using GridSearchCV and RandomizedSearchCV to optimize performance.

Model evaluation metrics included accuracy, runtime, feature importance, and confusion matrices to give a comprehensive view of each model's predictive power. The results showed varying accuracy levels, with bagging classifiers improving performance for some models. Overall, the project demonstrated the potential of machine learning for classifying legal U.S. tender and highlighted optimization strategies to enhance prediction accuracy.

Objective

The objective of this project is to build and evaluate machine learning models to predict whether a coin is legal U.S. tender or not based on its physical characteristics. The dataset includes 360 coins, which are split evenly between legal U.S. tender and foreign tender. The project involves applying data preprocessing techniques, selecting and optimizing predictive models, and evaluating them based on accuracy, runtime, feature importance, and confusion matrices. The goal is to show how machine learning can be applied to classification tasks and explore the effectiveness of various models and optimization methods in predicting whether a coin is legal U.S. tender or not.

Introduction

Coin collecting has always been a fun hobby in my family, with a collection that includes coins from all over the world. In this project, I applied predictive modeling to figure out if a coin is legal U.S. tender or not. The dataset contained coins with different physical properties, and by analyzing these features, I worked to predict whether a coin is U.S. currency or foreign. My goal was to understand the characteristics that determine if a coin is legal U.S. tender, and if not, I would consider it foreign tender. Using machine learning techniques, I aimed to make accurate predictions and gain insights into how different features impact the classification.

Methodology

Dataset:

The dataset for this project was a self-created collection of coins, containing a total of 360 coins, split evenly between 180 legal U.S. tender coins and 180 foreign tender coins. Each coin was individually measured and recorded based on key physical properties such as diameter, thickness, weight, edge type, magnetism, and main metal composition. The goal was to use these features to predict whether a coin is legal U.S. tender or foreign tender. This dataset provided the foundation for applying machine learning techniques to classify coins based on their physical attributes.

Coin Properties:

1. **Diameter:** Measured in millimeters using a dial caliper.
2. **Thickness:** Also measured in millimeters with the same dial caliper.
3. **Weight:** Recorded in grams using a precision scale.
4. **Main Metal Composition:** The primary metal content of each coin was determined by researching online sources such as numista.com and using Google Lens, focusing on the metal that made up the highest percentage of the coin.
5. **Edge Type:** Identified through visual and tactile examination, coins were categorized based on their edge type (e.g., smooth, reeded, etc.).
6. **Magnetism:** A magnet was used to determine whether the coin was magnetic.
7. **Legal U.S. Tender:** The target variable, representing whether the coin is recognized as legal tender in the United States. This information was labeled based on personal knowledge of U.S. currency.

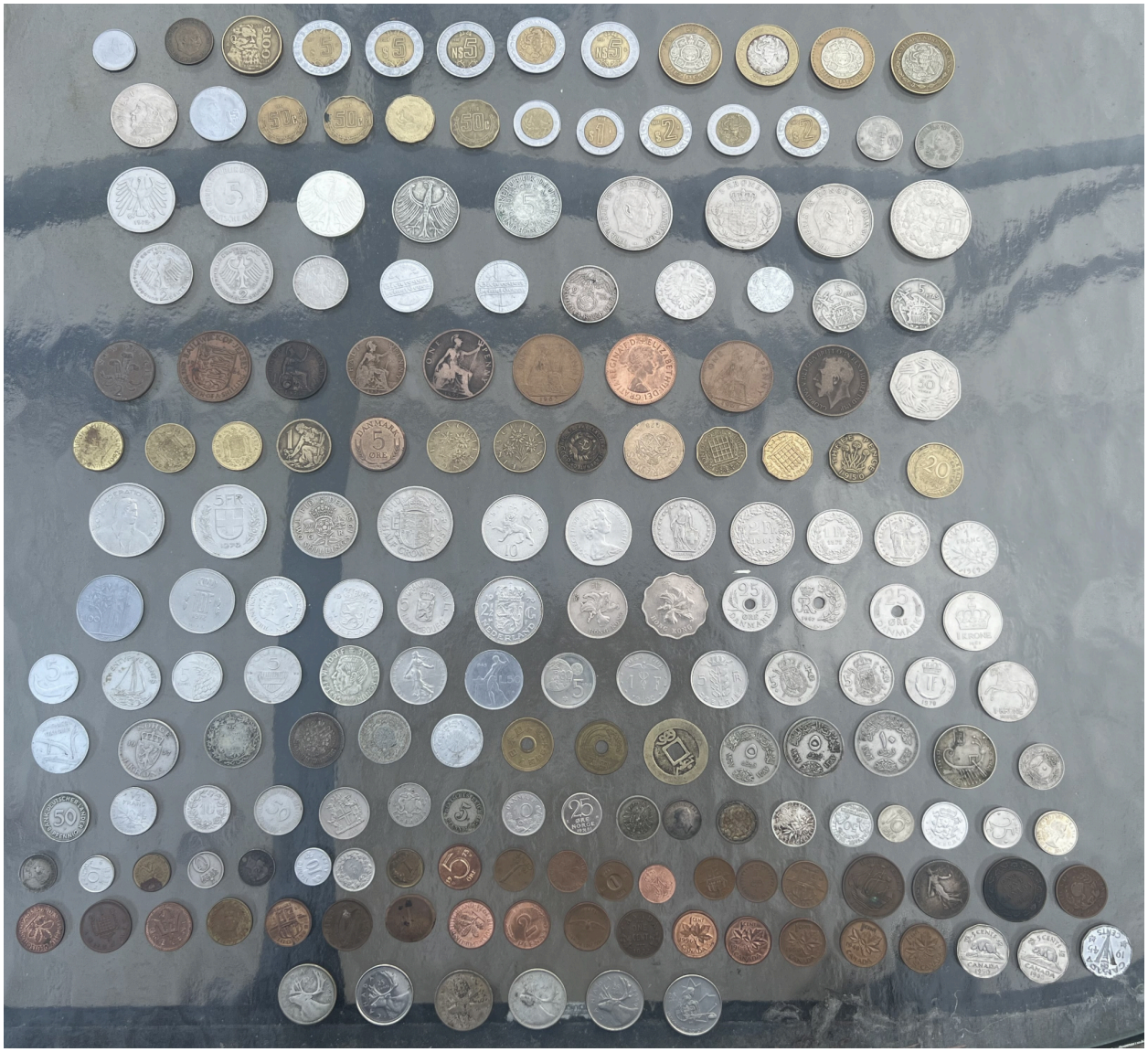
Methodology Cont.

Coin Photographs:

U.S. Tender: The 180 coins identified as legal U.S. tender.

Methodology Cont.

Foreign Tender: The 180 coins that are foreign tender.



Methodology Cont.

U.S. Legal Tender Coin Breakdown by Type

1. **1 Dollar Coins:**
    - George Washington: 7 coins
    - Sacagawea Dollar: 7 coins
    - Susan B. Anthony Dollar: 8 coins
    - Eisenhower Bicentennial Dollar: 2 coins
    - Eisenhower Dollar: 2 coins
    - Morgan Dollar: 4 coins
2. **½ Dollar Coins (Kennedy Half Dollar):**
    - 40% Silver: 7 coins
    - 90% Silver: 8 coins
    - Bicentennial: 8 coins
    - Standard: 7 coins
3. **¼ Dollar Coins:**
    - Washington Quarter (Bicentennial): 10 coins
    - Washington Quarter: 10 coins
    - Washington Silver Quarter: 10 coins

Methodology Cont.

4. **1 Dime Coins:**
    - Roosevelt Dime: 10 coins
    - Mercury Dime: 10 coins
    - Roosevelt Silver Dime: 10 coins
5. **5 Cents Coins:**
    - Jefferson Nickel: 21 coins
    - Wartime Jefferson Nickel: 9 coins
6. **1 Cent Coins:**
    - Lincoln Cent - Shield Reverse: 10 coins
    - Lincoln Cent - Wheat Ears Reverse: 10 coins
    - Steel Cent: 10 coins

The data collection process took several days, with each coin being meticulously measured and recorded. The dataset was complete with no missing values, and all variables were carefully preprocessed before model training.

Methodology Cont.

Statistical Overview of the Dataset:

1. **Diameter:**
    - Min: 14.89 mm
    - Max: 38.33 mm
    - Mean: 23.35 mm
2. **Thickness:**
    - Min: 0.71 mm
    - Max: 2.98 mm
    - Mean: 1.70 mm
3. **Weight:**
    - Min: 0.457 g
    - Max: 26.741 g
    - Mean: 6.05 g
4. **Edge Type:**
    - Reeded: 177 coins
    - Plain: 163 coins
    - Lettered: 19 coins
    - Decorated: 1 coin

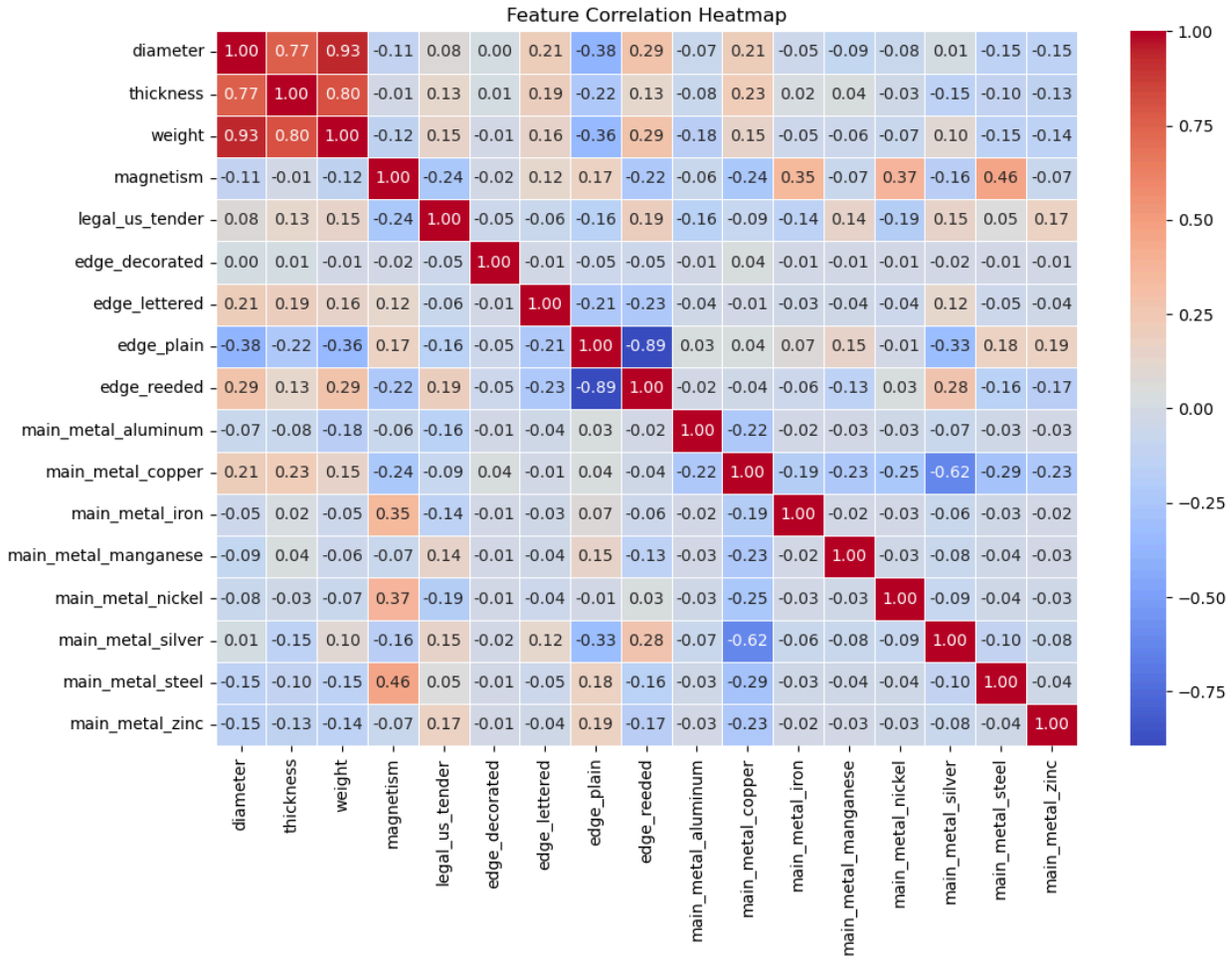
Methodology Cont.

5. **Main Metal:**
    - Copper: 233 coins
    - Silver: 63 coins
    - Steel: 16 coins
    - Nickel: 12 coins
    - Zinc: 10 coins
    - Manganese: 10 coins
    - Aluminum: 9 coins
    - Iron: 7 coins
6. **Magnetism:**
    - Non-Magnetic: 310 coins
    - Magnetic: 50 coins
7. **Legal U.S. Tender (Target Variable):**
    - Legal U.S. Tender: 180 coins
    - Non-Legal U.S. Tender (Foreign Tender): 180 coins

The dataset is well-balanced, containing an equal number of legal U.S. tender coins and foreign tender coins. This balanced distribution ensures that both categories are equally represented, allowing for fair and unbiased prediction modeling.

Methodology Cont.

Feature Correlation Heatmap:



The feature correlation heatmap provides insight into the relationships between numerical variables in the dataset. It visually represents the strength and direction of correlations, helping to identify highly correlated features that may impact model performance. Strong positive correlations (close to 1) indicate that two features increase together, while strong negative correlations (close to -1) suggest an inverse relationship. Features with little to no correlation (close to 0) are largely independent.

Methodology Cont.

Correlation Comparison:

1. **Strong Positive Correlations:**
    - diameter & weight = 0.930804
    - thickness & weight = 0.804688
    - diameter & thickness = 0.766446
    - magnetism & main_metal_steel = 0.459051
    - magnetism & main_metal_nickel = 0.372885
    - magnetism & main_metal_iron = 0.350637
    - diameter & edge_reeded = 0.287281
    - weight & edge_reeded = 0.286180
    - edge_reeded & main_metal_silver = 0.278206
    - thickness & main_metal_copper = 0.227050
    - diameter & main_metal_copper = 0.214223
    - diameter & edge_lettered = 0.208720

The highest correlations were observed between diameter, thickness, and weight, with diameter and weight having the strongest correlation at 0.9308. This relationship is expected, as larger coins tend to be both heavier and thicker. Magnetism also showed positive correlations with specific metal compositions, particularly steel (0.4591), nickel (0.3729), and iron (0.3506), which are known for their magnetic properties. Additionally, a moderate correlation was found between reeded edges and both weight (0.2862) and diameter (0.2873), indicating that larger and heavier coins are more likely to have a reeded edge.

Methodology Cont.

2. **Strong Negative Correlations:**

- edge_plain & edge_reeded = -0.894585
- main_metal_copper & main_metal_silver = -0.623833
- diameter & edge_plain = -0.382462
- weight & edge_plain = -0.359826
- edge_plain & main_metal_silver = -0.330820
- main_metal_copper & main_metal_steel = -0.292117
- main_metal_copper & main_metal_nickel = -0.251523
- magnetism & main_metal_copper = -0.241404
- magnetism & legal_us_tender = -0.240966
- edge_lettered & edge_reeded = -0.232146
- main_metal_copper & main_metal_zinc = -0.228951
- main_metal_copper & main_metal_manganese = -0.228951
- magnetism & edge_reeded = -0.218238
- thickness & edge_plain = -0.217211
- main_metal_aluminum & main_metal_copper = -0.216892
- edge_lettered & edge_plain = -0.214714

The most significant negative correlation was between plain and reeded edges (-0.8946), which is expected since a coin can only have one edge type. Copper and silver content also showed a notable negative correlation (-0.6238), indicating that coins are typically made of one or the other but rarely both in significant amounts. Edge type and size-related features exhibited negative correlations, with plain edges being more common on smaller, lighter coins, as seen in the correlations between diameter and plain edges (-0.3825) and weight and plain edges (-0.3598). Additionally, magnetism had a weak negative correlation with legal U.S. tender (-0.2410), suggesting that U.S. coins are generally non-magnetic compared to foreign coins.

One of the most important findings was that no single feature had a strong correlation with "legal_us_tender", meaning that no individual attribute alone could determine if a coin was U.S. legal tender. This reinforces the need for machine learning models, as they can identify complex patterns and interactions between multiple features that simple correlation analysis cannot capture.

Methodology Cont.

Preprocessing:

Before training machine learning models, several preprocessing steps were performed on the dataset to ensure that the features were properly formatted and scaled for optimal model performance. The following preprocessing tasks were completed:

1. **Scaling:**

The features "weight," "diameter," and "thickness" were scaled using the MinMaxScaler. This scaling method was chosen to normalize the values of these features to a range between 0 and 1. MinMax scaling helps prevent features with larger numerical ranges from disproportionately influencing model training, ensuring that all numerical features contribute equally to the model.

2. **Encoding Categorical Variables:**

The "magnetism" and "legal_us_tender" columns were categorical variables that needed to be transformed into numerical format. For this purpose, a LabelEncoder was applied. This encoder converted the categories into numerical labels, with "magnetism" indicating whether a coin is magnetic (binary: 1 for magnetic, 0 for not magnetic) and "legal_us_tender" indicating whether a coin is recognized as legal U.S. tender (binary: 1 for legal, 0 for foreign).

3. **Creating Dummy Variables:**

The columns "edge" and "main_metal" contained categorical data that needed to be converted into a format that machine learning algorithms could use. To achieve this, one-hot encoding (via pd.get_dummies) was applied to these columns. This process created separate binary indicator columns for each unique category within the "edge" and "main_metal" variables. For instance, if the "edge" column included values like "smooth" and "reeded," new columns for each type ("edge_smooth" and "edge_reeded") were created, with values of 1 or 0 indicating the presence of each category in the original feature.

4. **Handling Missing Data:**

The dataset did not contain any missing values, so no imputation or removal of rows was necessary. Each feature was fully populated with valid data.

5. **Splitting the Dataset:**

The dataset was split into training and testing sets, with a 70/30 split. This means that 70% of the data was used for training the models, while 30% was reserved for evaluating the performance of the models on unseen data.

These preprocessing steps were essential for ensuring that the dataset was appropriately structured and ready for model training. They helped standardize the input features, transforming them into a numerical format suitable for machine learning algorithms.

Methodology Cont.

Models Evaluated:

1. **K-Nearest Neighbors (KNN):** A non-parametric classifier based on the proximity of data points.
2. **Decision Tree (DT):** A tree-structured model that splits data based on feature importance.
3. **Random Forest (RF):** An ensemble of decision trees used for classification tasks.
4. **Perceptron:** A simple linear classifier that uses gradient descent.
5. **Logistic Regression (LR):** A regression-based classifier for binary classification problems.

Bagging classifiers were used with each of the five models to investigate if they could potentially improve accuracy by reducing variance in predictions.

Model Parameters:

For each classifier, I chose the parameter ranges based on the impact they could have on the model's performance and how they aligned with the characteristics of the dataset. I selected the ranges I believed were most relevant for optimizing each model and maximizing prediction accuracy. Here's a breakdown of the models and their parameter ranges:

1. **Bagging Classifier**
    - **n_estimators:** The number of base estimators in the ensemble influences model complexity and variance. I selected 5, 10, 25, 50, 100, 200, and 400 to cover a broad range, from smaller ensembles that may underfit to larger ensembles that might reduce variance but increase computation time.
    - **max_samples:** This determines the proportion of the dataset used to train each base estimator. By selecting 0.5, 0.75, and 1.0, I aimed to examine how different levels of data subsampling impact model variance and bias.
    - **bootstrap:** Sampling with replacement can help reduce overfitting in ensemble learning. By including both True and False, I evaluated whether bootstrapping improves generalization.
    - **bootstrap_features:** Similar to bootstrap, this controls whether feature sampling is performed with replacement. Testing both settings helps assess whether additional randomness improves model performance.

Methodology Cont.

2. **K-Nearest Neighbors (KNN)**
    - **n_neighbors:** Controls the number of nearest neighbors used for classification. I tested 3, 5, 7, 9, and 11 to observe the balance between model flexibility and stability, ensuring neither underfitting nor excessive sensitivity to noise.
    - **weights:** Determines how neighbors contribute to classification. I included 'uniform' (equal weight for all neighbors) and 'distance' (closer neighbors have higher influence) to compare the impact of weighting on prediction accuracy.
    - **algorithm:** Specifies the method used for nearest neighbor searches. I tested 'auto', 'ball_tree', 'kd_tree', and 'brute' to determine which search strategy performed best for my dataset.
    - **leaf_size:** Affects the efficiency of tree-based neighbor searches. I explored values from 5 to 75 to balance computational speed and accuracy, ensuring the algorithm runs efficiently while maintaining performance.
    - **p:** Defines the distance metric used for calculations. I included 1 (Manhattan distance) and 2 (Euclidean distance) to compare their effects on classification accuracy and model behavior.

3. **Decision Tree (DT)**

- **criterion:** Determines the function used to measure the quality of splits. I included 'gini' and 'entropy' to compare how different impurity measures affect decision tree performance.
- **max_depth:** Controls the maximum depth of the tree, influencing complexity and overfitting. I tested 5, 10, 20, 40, and None to observe the impact of tree depth on model accuracy.
- **min_samples_split:** Specifies the minimum number of samples required to split a node. I tested 2, 5, 10, and 20 to examine how different split thresholds influence tree growth and generalization.
- **min_samples_leaf:** Sets the minimum number of samples a leaf node must have. I used 1, 2, 5, and 10 to analyze how leaf size affects model stability and performance.
- **max_features:** Limits the number of features considered for each split. I included None (all features), 'sqrt' (square root of total features), and 'log2' (logarithm of total features) to evaluate how feature selection impacts decision boundaries.
- **max_leaf_nodes:** Restricts the total number of leaf nodes in the tree. I tested None, 10, 50, and 100 to explore how pruning affects tree complexity and accuracy.
- **splitter:** Determines how splits are chosen at each node. I compared 'best' (choosing the best split) and 'random' (randomly selecting a split) to see how randomness influences model performance.
- **class_weight:** Adjusts weights assigned to different classes to handle imbalanced data. I tested None (no weighting) and 'balanced' (adjusting weights inversely proportional to class frequencies) to assess the impact on classification results.

Methodology Cont.

4. **Random Forest (RF)**

- **n_estimators:** The number of trees in the forest affects both model accuracy and computation time. I tested 5, 10, 25, 50, and 75 to explore the trade-off between performance and computation cost.
- **criterion:** Defines the function used to evaluate the quality of splits in each tree. I compared 'gini' and 'entropy' to assess how each impurity measure influences the overall performance of the random forest.
- **max_depth:** Controls the maximum depth of each tree, impacting model complexity and overfitting. I tested 5, 10, 20, and None to investigate how tree depth affects generalization and overfitting.
- **min_samples_split:** Specifies the minimum number of samples needed to split a node. I experimented with 2, 5, 10, and 20 to see how different values of node splitting influence the model's complexity and performance.

Methodology Cont.

- **min_samples_leaf:** Defines the minimum number of samples required in a leaf node. I tested 1, 2, 5, and 10 to examine how pruning nodes with fewer samples can reduce overfitting and improve model stability.
- **max_features:** Controls the number of features considered for each split. I tested None, 'sqrt', and 'log2' to evaluate the impact of different feature selection strategies on model diversity and performance.
- **bootstrap:** Whether to sample with replacement influences variance and overfitting. I tested both True and False to assess how bootstrapping impacts model performance and its ability to generalize to new data.
- **class_weight:** Adjusts the weight assigned to each class to address class imbalance. I compared None (no weight adjustment) and 'balanced' (weights inversely proportional to class frequencies) to examine the effect on accuracy and fairness in imbalanced datasets.

Methodology Cont.

5. **Perceptron**

- **penalty:** Specifies the regularization technique to control overfitting. I tested 'l1', 'l2', and 'elasticnet' to determine how each regularization method affects model generalization and complexity.
- **alpha:** Controls the strength of regularization. I varied it from 0.0001 to 1 to explore how different levels of regularization influence model performance and prevent overfitting.
- **max_iter:** Defines the maximum number of iterations for optimization. I tested values from 500 to 7500 to ensure the model converges and achieves optimal performance without unnecessary computations.
- **tol:** Determines the tolerance for stopping criteria. I experimented with values from 1e-5 to 1e-2 to fine-tune the convergence threshold and balance between training time and model accuracy.

Methodology Cont.

- **eta0:** The initial learning rate influences how quickly the model learns. I varied it from 0.001 to 1 to analyze its effect on training speed and final performance, ensuring neither too slow nor too fast convergence.
- **early_stopping:** Determines whether the model should stop training early when performance stops improving. I tested both True and False to assess the effect of early stopping on model accuracy and training time.
- **n_iter_no_change:** Specifies how many iterations without improvement before triggering early stopping. I explored values of 5, 10, 15, 20, and 25 to strike a balance between preventing overfitting and reducing training time.
- **validation_fraction:** The proportion of the training data used for validation when early stopping is enabled. I tested values of 0.1, 0.2, and 0.3 to see how validation set size influences the model's stopping criteria and overall performance.

Methodology Cont.

6. **Logistic Regression (LR)**

- **C:** Controls the regularization strength. I varied this parameter from 0.01 to 50 to assess how different levels of regularization impact model performance and help prevent overfitting.
- **solver:** Specifies the optimization algorithm used for fitting the model. I tested 'newton-cg', 'lbfgs', and 'saga' to determine which solver provided the best convergence speed and accuracy for my dataset.
- **max_iter:** Sets the maximum number of iterations for the optimization process. I tested values ranging from 100 to 3000 to ensure the model has enough iterations to converge, especially with more complex datasets.
- **tol:** Adjusts the tolerance for optimization stopping criteria. I experimented with tolerance values from 1e-4 to 1e-1 to fine-tune the stopping condition and ensure the model converges without excessive computation.
- **class_weight:** This parameter was varied between None and 'balanced' to address class imbalances. 'Balanced' adjusts weights inversely proportional to class frequencies, helping the model handle imbalanced datasets more effectively.
- **intercept_scaling:** Determines the scaling of the intercept term in the decision function. I tested values of 1, 5, and 10 to understand how the intercept scaling affects convergence and model performance.
- **warm_start:** Whether to reuse the previous solution when fitting the model. I tested both True and False to evaluate if reusing the previous solution speeds up training and improves model efficiency without compromising accuracy.

Methodology Cont.

These ranges were chosen based on observational experience of how each parameter impacts model behavior, ensuring that I explored both potential extremes and more balanced options for each classifier. By experimenting within these ranges, I was able to fine-tune each model and identify the optimal parameters that contributed to the best performance on the dataset.

Methodology Cont.

Procedure:

1. **Data Preprocessing**

Before model training, the dataset underwent several preprocessing steps to ensure all features were appropriately formatted and scaled:

- The features "weight," "diameter," and "thickness" were scaled using the MinMaxScaler to normalize the values into a range between 0 and 1, preventing any feature from dominating model performance due to scale differences.
- The "magnetism" and "legal_us_tender" columns were transformed into numerical labels using LabelEncoder. This step enabled the models to interpret the categorical data correctly.
- The "edge" and "main_metal" categorical variables were transformed into binary indicator columns using one-hot encoding via pd.get_dummies.

These preprocessing steps ensured that all features were in a suitable format for training machine learning models.

Methodology Cont.

2. Data Splitting

Data splitting ensured proper model training and evaluation by dividing the dataset and applying cross-validation.

- The dataset was split into training (70%) and testing (30%) subsets.
- K-Fold cross-validation (k=10) was used for model evaluation.

A 70/30 train-test split was chosen to maintain a balance between training the model on enough data for learning while preserving a sufficient portion for evaluating performance on unseen data. Allocating 70% of the dataset for training allowed the models to learn meaningful patterns, while the 30% test set provided a robust evaluation of generalization ability.

For cross-validation, 10-fold cross-validation was selected to improve the reliability of performance estimates. By dividing the training data into 10 subsets, each model was trained and validated multiple times, reducing the impact of data variability. This approach ensured that every instance in the dataset was used for both training and validation, leading to a more stable and accurate evaluation of model performance. The choice of k=10 provided a good balance between computational efficiency and a thorough assessment of each model's predictive capabilities.

Methodology Cont.

3. Model Training & Evaluation

To systematically assess model performance, each classifier was first trained with default hyperparameters to establish a baseline. Accuracy was then evaluated by comparing predictions to actual labels. Additionally, Bagging Classifiers were applied to reduce variance and improve model stability by aggregating predictions from multiple base learners.

- Each model was trained using default hyperparameters to serve as a baseline for evaluation.
- Predictions were made on the test dataset, and accuracy was computed by comparing predicted labels to true labels.
- Bagging Classifiers were applied to each model to assess performance improvements through variance reduction and enhanced stability.
- The bagging technique helped improve predictive power by combining the predictions of multiple base learners.

By applying these techniques, the models' performance was systematically evaluated, with bagging providing an additional method to enhance accuracy and reduce overfitting.

Methodology Cont.

4. Hyperparameter Optimization

To maximize model performance, GridSearchCV and RandomizedSearchCV were used to identify the best hyperparameters for each model. After optimizing the base models, the Bagging Classifiers were also fine-tuned by selecting the best hyperparameter combinations, further improving predictive accuracy and stability.

- GridSearchCV performed an exhaustive search to find the optimal hyperparameters for each model.
- RandomizedSearchCV was used to explore a wider range of hyperparameters in a more computationally efficient manner.
- Once base model hyperparameters were optimized, Bagging Classifiers were fine-tuned to achieve the best combination of parameters.

By systematically optimizing hyperparameters for both base models and their bagging classifiers, model performance was refined, ensuring the best possible accuracy and generalization.

Methodology Cont.

5. Final Model Evaluation

After optimizing the models through hyperparameter tuning, the final evaluation was performed using the test dataset.

- The models were compared based on their accuracy in predicting legal U.S. tender coins. The model with the highest accuracy was considered the best-performing model for the task.
- The performance of the base models and the bagging classifiers was assessed to determine whether the bagging technique improved accuracy.

The final evaluation concluded by selecting the model and its potential parameters that delivered the highest accuracy from the coin dataset as the most effective for the task.

Methodology Cont.

Metrics:

To evaluate model performance comprehensively, multiple metrics were recorded, including accuracy, runtime, confusion matrices, and feature importance (where applicable). These metrics provided insights into both predictive power and computational efficiency.

- **Accuracy:** The percentage of correctly classified instances, serving as the primary performance measure.
- **Runtime:** The time taken to train and evaluate each model, offering insight into computational efficiency.
- **Confusion Matrix:** A visualization of classification results, detailing True Positives, False Positives, True Negatives, and False Negatives to better understand model behavior.
- **Feature Importance:** For Decision Tree and Random Forest, key features influencing predictions were identified to interpret their impact.

By considering these metrics, the balance between accuracy, speed, and interpretability was assessed, helping to determine the most effective model for the task.

Results

Predictive Model's Results:

This section presents the results of each machine learning model applied to the dataset. For each classifier, performance metrics such as runtime and accuracy are provided under different configurations, including default parameters, optimized parameters found via GridSearchCV and RandomizedSearchCV, and the application of bagging techniques. These results help assess the effectiveness of each model in predicting the target outcome.

1. **K-Nearest Neighbor (KNN)**
    - KNN Runtime Using Default KNN Params: 0.20659 seconds
    - KNN Accuracy Score Using Default KNN Params: 87.963%
    - Bag Runtime Using Default Bag Params w/ Default KNN Params: 1.20243 seconds
    - Bag Accuracy Score Using Default Bag Params w/ Default KNN Params: 89.815%
    - Best KNN Params According To GridSearchCV: {'algorithm': 'auto', 'leaf_size': 5, 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
    - KNN Runtime Using Best KNN Params According To GridSearchCV: 3.18013 seconds
    - KNN Accuracy Score Using Best KNN Params According To GridSearchCV: 94.444%
    - Best KNN Params According To RandomizedSearchCV: {'weights': 'distance', 'p': 1, 'n_neighbors': 3, 'leaf_size': 5, 'algorithm': 'kd_tree'}

Results Cont.

- KNN Runtime Using Best KNN Params According To RandomizedSearchCV: 0.05234 seconds
- KNN Accuracy Score Using Best KNN Params According To RandomizedSearchCV: 94.444%
- Best Bag Params According To GridSearchCV: {'bootstrap': False, 'bootstrap_features': True, 'max_samples': 1.0, 'n_estimators': 200}
- Bag Runtime Using Best Bag Params w/ Best KNN Params According To GridSearchCV: 6.14233 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best KNN Params According To GridSearchCV: 95.370%
- Best Bag Params According To RandomizedSearchCV: {'n_estimators': 400, 'max_samples': 1.0, 'bootstrap_features': True, 'bootstrap': True}
- Bag Runtime Using Best Bag Params w/ Best KNN Params According To RandomizedSearchCV: 4.38098 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best KNN Params According To RandomizedSearchCV: 94.444%

Results Cont.

2. **Decision Tree (DT)**

- DT Runtime Using Default DT Params: 0.00300 seconds
- DT Accuracy Score Using Default DT Params: 86.111%
- Bag Runtime Using Default Bag Params w/ Default DT Params: 1.98918 seconds
- Bag Accuracy Score Using Default Bag Params w/ Default DT Params: 91.667%
- Best DT Params According To GridSearchCV: {'class_weight': None, 'criterion': 'entropy', 'max_depth': 20, 'max_features': None, 'max_leaf_nodes': 100, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
- DT Runtime Using Best DT Params According To GridSearchCV: 14.72791 seconds
- DT Accuracy Score Using Best DT Params According To GridSearchCV: 94.444%
- Best DT Params According To RandomizedSearchCV: {'splitter': 'random', 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_leaf_nodes': None, 'max_features': 'log2', 'max_depth': None, 'criterion': 'gini', 'class_weight': 'balanced'}

Results Cont.

- DT Runtime Using Best DT Params According To RandomizedSearchCV: 0.07309 seconds
- DT Accuracy Score Using Best DT Params According To RandomizedSearchCV: 91.667%
- Best Bag Params According To GridSearchCV: {'bootstrap': False, 'bootstrap_features': True, 'max_samples': 1.0, 'n_estimators': 100}
- Bag Runtime Using Best Bag Params w/ Best DT Params According To GridSearchCV: 7.46132 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best DT Params According To GridSearchCV: 98.148%
- Best Bag Params According To RandomizedSearchCV: {'n_estimators': 200, 'max_samples': 0.75, 'bootstrap_features': True, 'bootstrap': True}
- Bag Runtime Using Best Bag Params w/ Best DT Params According To RandomizedSearchCV: 2.43139 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best DT Params According To RandomizedSearchCV: 97.222%

Results Cont.

3. **Random Forest (RF)**

- RF Runtime Using Default RF Params: 0.06175 seconds
- RF Accuracy Score Using Default RF Params: 97.222%
- Bag Runtime Using Default Bag Params w/ Default RF Params: 2.11494 seconds
- Bag Accuracy Score Using Default Bag Params w/ Default RF Params: 94.444%
- Best RF Params According To GridSearchCV: {'bootstrap': False, 'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
- RF Runtime Using Best RF Params According To GridSearchCV: 457.57454 seconds
- RF Accuracy Score Using Best RF Params According To GridSearchCV: 94.444%
- Best RF Params According To RandomizedSearchCV: {'n_estimators': 25, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 20, 'criterion': 'gini', 'class_weight': None, 'bootstrap': False}

Results Cont.

- RF Runtime Using Best RF Params According To RandomizedSearchCV: 0.56273 seconds
- RF Accuracy Score Using Best RF Params According To RandomizedSearchCV: 96.296%
- Best Bag Params According To GridSearchCV: {'bootstrap': False, 'bootstrap_features': True, 'max_samples': 1.0, 'n_estimators': 100}
- Bag Runtime Using Best Bag Params w/ Best RF Params According To GridSearchCV: 191.83544 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best RF Params According To GridSearchCV: 95.370%
- Best Bag Params According To RandomizedSearchCV: {'n_estimators': 400, 'max_samples': 1.0, 'bootstrap_features': False, 'bootstrap': False}
- Bag Runtime Using Best Bag Params w/ Best RF Params According To RandomizedSearchCV: 94.80024 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best RF Params According To RandomizedSearchCV: 97.222%

Results Cont.

4. **Perceptron**

- Perceptron Runtime Using Default Perceptron Params: 0.00100 seconds
- Perceptron Accuracy Score Using Default Perceptron Params: 72.222%
- Bag Runtime Using Default Bag Params w/ Default Perceptron Params: 1.91110 seconds
- Bag Accuracy Score Using Default Bag Params w/ Default Perceptron Params: 73.148%
- Best Perceptron Params According To GridSearchCV: {'alpha': 0.001, 'early_stopping': True, 'eta0': 1, 'max_iter': 500, 'n_iter_no_change': 25, 'penalty': None, 'tol': 1e-05, 'validation_fraction': 0.1}
- Perceptron Runtime Using Best Perceptron Params According To GridSearchCV: 154.23980 seconds
- Perceptron Accuracy Score Using Best Perceptron Params According To GridSearchCV: 69.444%
- Best Perceptron Params According To RandomizedSearchCV: {'validation_fraction': 0.3, 'tol': 1e-05, 'penalty': 'l2', 'n_iter_no_change': 20, 'max_iter': 500, 'eta0': 0.01, 'early_stopping': True, 'alpha': 0.0001}

Results Cont.

- Perceptron Runtime Using Best Perceptron Params According To RandomizedSearchCV: 0.09692 seconds
- Perceptron Accuracy Score Using Best Perceptron Params According To RandomizedSearchCV: 63.889%
- Best Bag Params According To GridSearchCV: {'bootstrap': False, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 25}
- Bag Runtime Using Best Bag Params w/ Best Perceptron Params According To GridSearchCV: 55.96688 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best Perceptron Params According To GridSearchCV: 74.074%
- Best Bag Params According To RandomizedSearchCV: {'n_estimators': 200, 'max_samples': 0.75, 'bootstrap_features': False, 'bootstrap': True}
- Bag Runtime Using Best Bag Params w/ Best Perceptron Params According To RandomizedSearchCV: 10.01814 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best Perceptron Params According To RandomizedSearchCV: 73.148%

Results Cont.

5. **Logistic Regression (LR)**

- LR Runtime Using Default LR Params: 0.00200 seconds
- LR Accuracy Score Using Default LR Params: 67.593%
- Bag Runtime Using Default Bag Params w/ Default LR Params: 1.84968 seconds
- Bag Accuracy Score Using Default Bag Params w/ Default LR Params: 69.444%
- Best LR Params According To GridSearchCV: {'C': 0.01, 'class_weight': None, 'intercept_scaling': 1, 'max_iter': 100, 'solver': 'newton-cg', 'tol': 0.1, 'warm_start': True}
- LR Runtime Using Best LR Params According To GridSearchCV: 11.94676 seconds
- LR Accuracy Score Using Best LR Params According To GridSearchCV: 55.556%
- Best LR Params According To RandomizedSearchCV: {'warm_start': False, 'tol': 0.1, 'solver': 'newton-cg', 'max_iter': 100, 'intercept_scaling': 10, 'class_weight': 'balanced', 'C': 50}

Results Cont.

- LR Runtime Using Best LR Params According To RandomizedSearchCV: 0.06024 seconds
- LR Accuracy Score Using Best LR Params According To RandomizedSearchCV: 65.741%
- Best Bag Params According To GridSearchCV: {'bootstrap': False, 'bootstrap_features': True, 'max_samples': 1.0, 'n_estimators': 400}
- Bag Runtime Using Best Bag Params w/ Best LR Params According To GridSearchCV: 9.02997 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best LR Params According To GridSearchCV: 65.741%
- Best Bag Params According To RandomizedSearchCV: {'n_estimators': 400, 'max_samples': 0.5, 'bootstrap_features': True, 'bootstrap': True}
- Bag Runtime Using Best Bag Params w/ Best LR Params According To RandomizedSearchCV: 3.29277 seconds
- Bag Accuracy Score Using Best Bag Params w/ Best LR Params According To RandomizedSearchCV: 65.741%

Results Cont.

Model Performance Comparison:

This section presents the best results achieved by each machine learning model on the dataset, highlighting both their highest accuracy and their fastest runtime. The highest accuracy result for each model corresponds to the best-performing configuration identified through hyperparameter tuning, while the fastest runtime represents the quickest execution using default parameters. Additionally, this section includes the confusion matrices for each model to illustrate classification performance, as well as feature importance graphs for the Decision Tree and Random Forest models.

Results Cont.

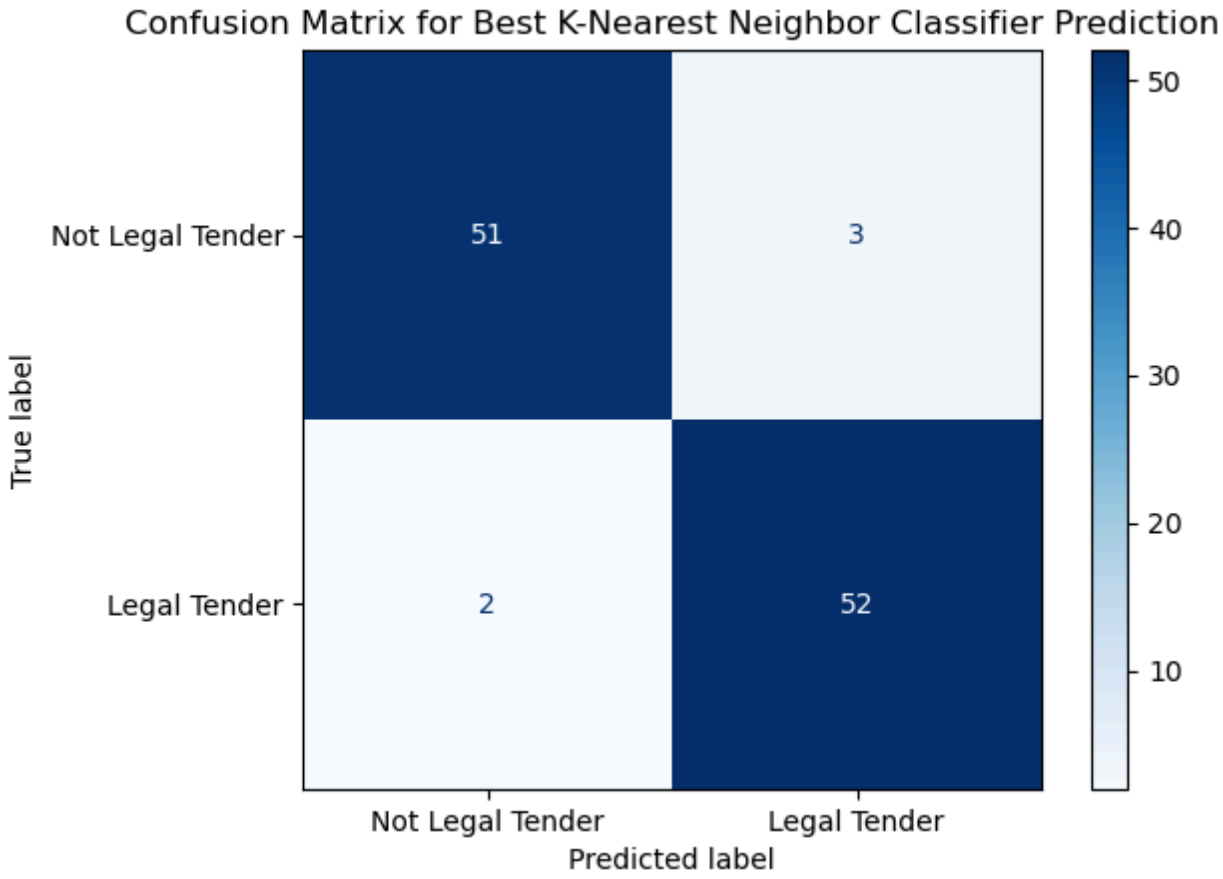
1. **K-Nearest Neighbors (KNN)**
    - **Highest Accuracy K-Nearest Neighbors (KNN) Prediction:** Bag Accuracy Score Using Best Bag Params w/ Best KNN Params According To GridSearchCV:
        - Runtime: 6.14233 seconds
        - Accuracy: 95.370%
    - **Fastest K-Nearest Neighbors (KNN) Prediction:** KNN Runtime Using Default KNN Params:
        - Runtime: 0.20659 seconds
        - Accuracy: 87.963%

For K-Nearest Neighbors (KNN), the best results were achieved with the following hyperparameters: n_neighbors set to 3, weights set to 'distance', algorithm set to 'auto', leaf_size set to 5, and p set to 1. This configuration resulted in an accuracy of 95.37% and took 6.14233 seconds to complete. The fastest KNN prediction, however, was obtained with the default parameters, which yielded an accuracy of 87.96% and completed in just 0.20659 seconds.

Results Cont.

- **K-Nearest Neighbors (KNN) Confusion Matrix:**

The confusion matrix offers insight into the model's classification accuracy, showing the distribution of true positives, true negatives, false positives, and false negatives.



False Positive (FP): 3

False Negative (FN): 2

Results Cont.

2. **Decision Tree (DT)**

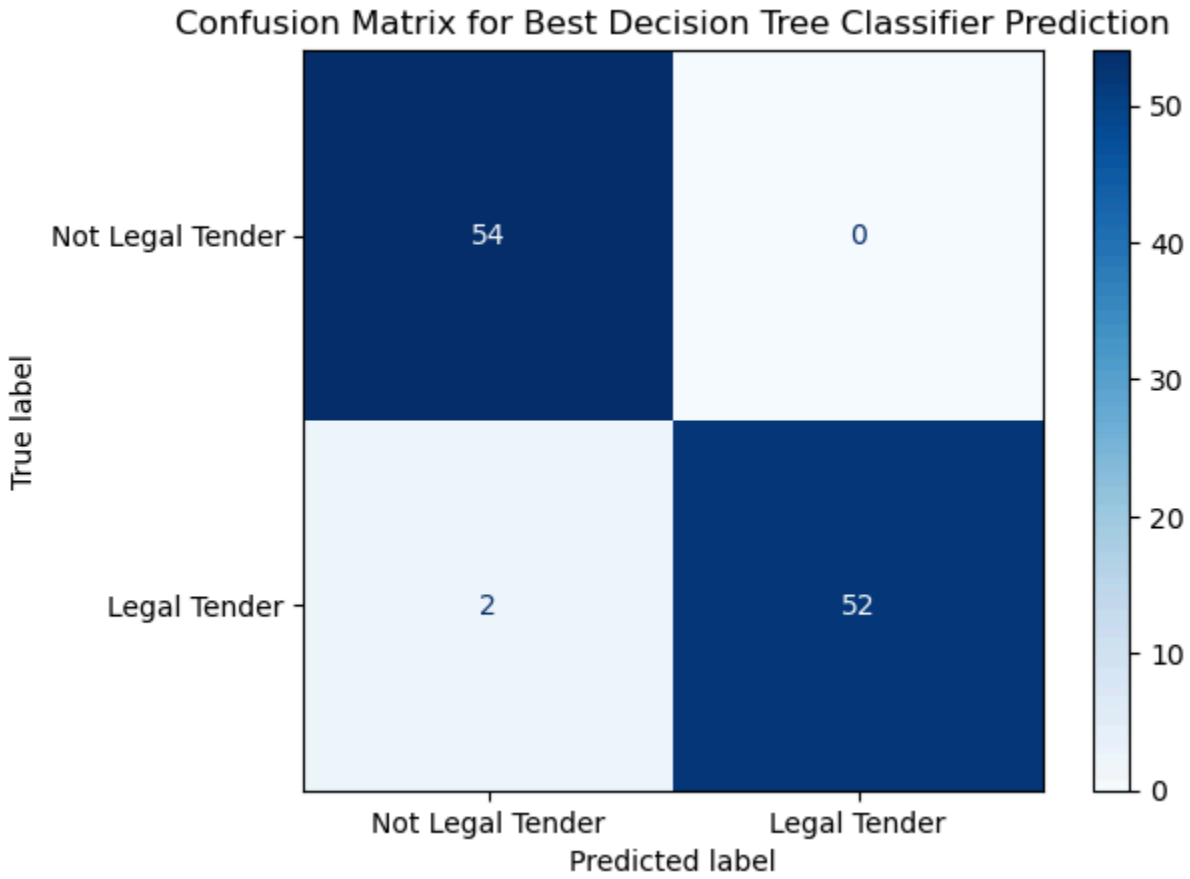
- **Highest Accuracy Decision Tree (DT) Prediction:** Bag Accuracy Score Using Best Bag Params w/ Best DT Params According To GridSearchCV
  - Runtime: 7.46132 seconds
  - Accuracy: 98.148%
- **Fastest Decision Tree (DT) Prediction:** DT Accuracy Score Using Default DT Params
  - Runtime: 0.003 seconds
  - Accuracy: 86.111%

The Decision Tree (DT) model performed exceptionally well, with the best accuracy of 98.15% when using the following hyperparameters: criterion set to 'entropy', max_depth set to 20, max_features set to None, max_leaf_nodes set to 100, min_samples_leaf set to 1, min_samples_split set to 2, splitter set to 'random', and class_weight set to None. The runtime for this model was 7.46132 seconds. Interestingly, the fastest DT prediction was completed in just 0.003 seconds with the default parameters, which resulted in an accuracy of 86.11%.

Results Cont.

- **Decision Tree (DT) Confusion Matrix:**

The confusion matrix offers insight into the model's classification accuracy, showing the distribution of true positives, true negatives, false positives, and false negatives.



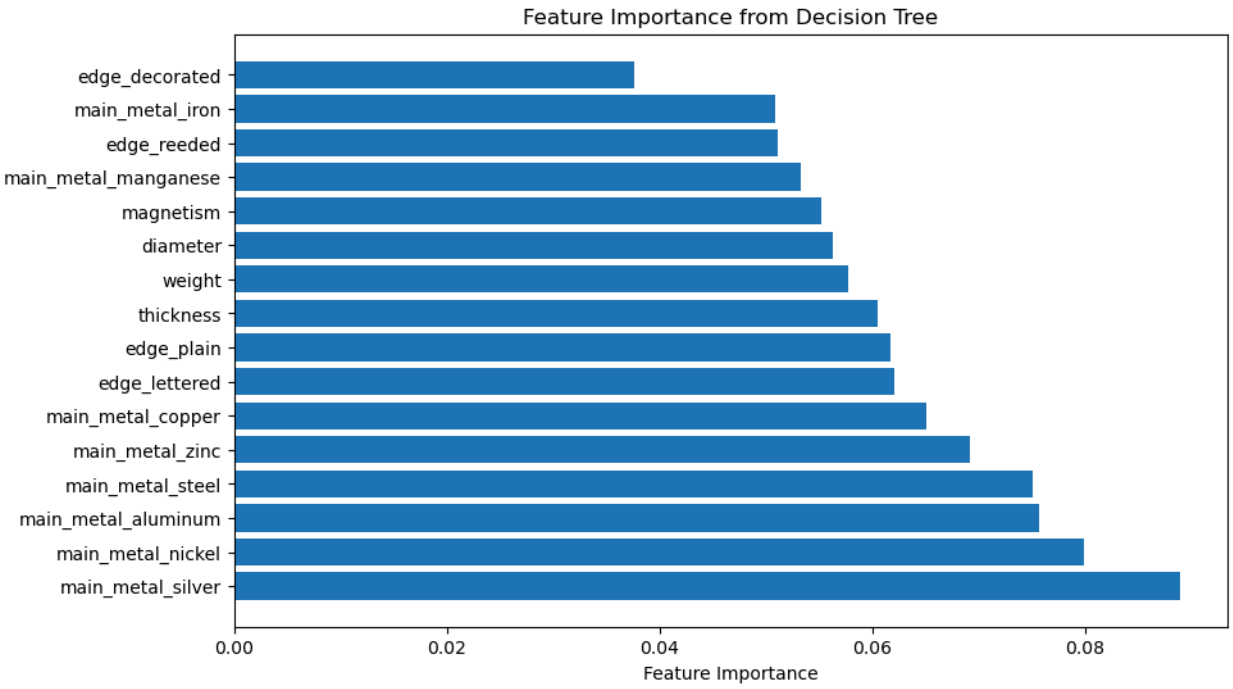
False Positive (FP): 0

False Negative (FN): 2

Results Cont.

- **Decision Tree (DT) Feature Importance Graph:**

The feature importance graph below visually represents how each feature influenced the model's classification.



The Decision Tree (DT) model assigns importance to each feature based on how much it reduces impurity when making splits. The percentages below represent the relative contribution of each feature to the model's decision-making.

Results Cont.

- **Decision Tree (DT) Feature Importance Comparison**

  1. **edge_decorated:** 3.763%
  2. **main_metal_iron:** 5.088%
  3. **edge_reeded:** 5.114%
  4. **main_metal_manganese:** 5.325%
  5. **magnetism:** 5.517%
  6. **diameter:** 5.631%
  7. **weight:** 5.773%
  8. **thickness:** 6.043%
  9. **edge_plain:** 6.17%
  10. **edge_lettered:** 6.208%
  11. **main_metal_copper:** 6.506%
  12. **main_metal_zinc:** 6.911%
  13. **main_metal_steel:** 7.503%
  14. **main_metal_aluminum:** 7.568%
  15. **main_metal_nickel:** 7.991%
  16. **main_metal_silver:** 8.888%

Results Cont.

The feature importance results from the Decision Tree (DT) model indicate which attributes had the greatest influence in predicting whether a coin is legal U.S. tender. The most significant feature was main_metal_silver (8.888%), suggesting that the presence of silver played a crucial role in classification. Other highly influential features included main_metal_nickel (7.991%), main_metal_aluminum (7.568%), and main_metal_steel (7.503%), highlighting the importance of a coin's primary metal composition in distinguishing legal U.S. tender from foreign coins.

Physical characteristics such as diameter (5.631%), weight (5.773%), and thickness (6.043%) also contributed notably to the model's decisions, reinforcing the role of a coin's size and mass in classification. Among edge types, edge_lettered (6.208%) and edge_plain (6.17%) had higher importance compared to edge_decorated (3.763%), suggesting that certain edge designs may be more indicative of U.S. coinage.

Results Cont.

3. **Random Forest (RF)**

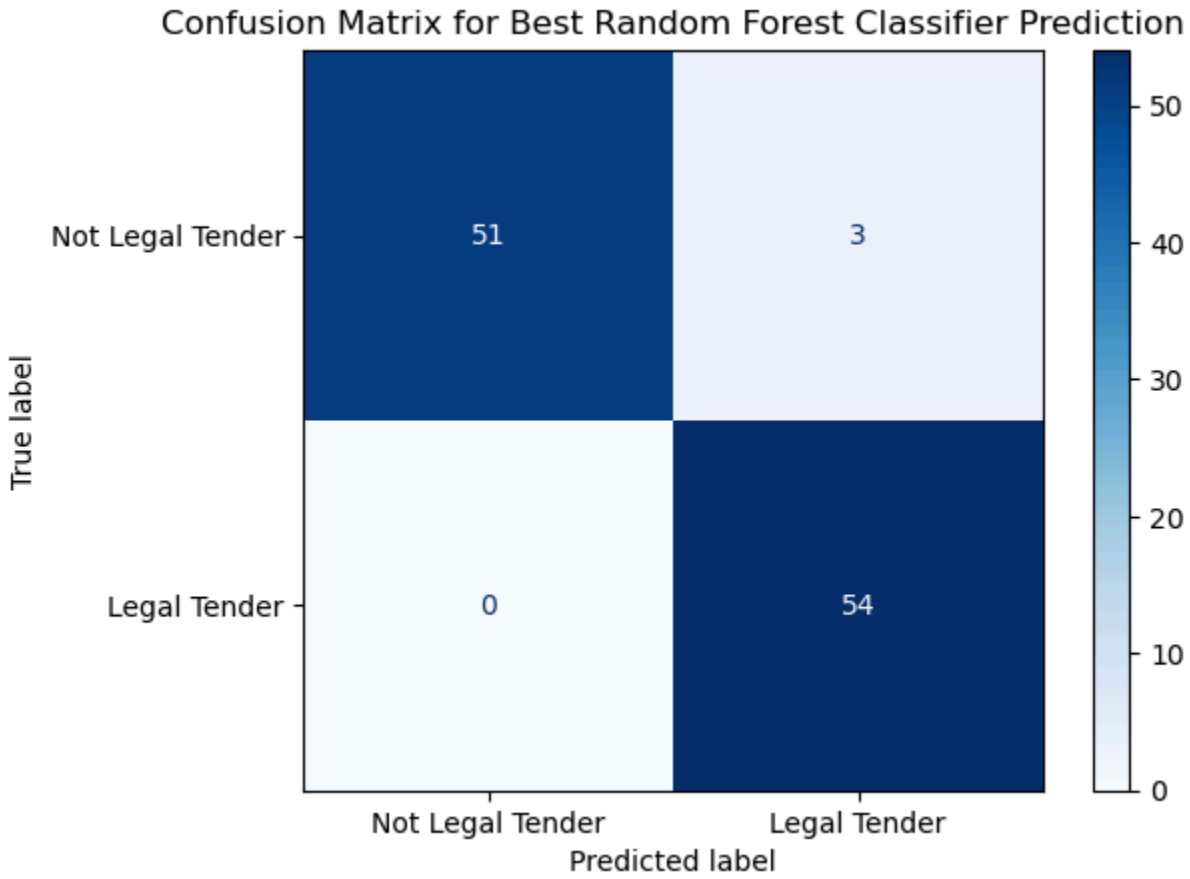
- **Highest Accuracy Random Forest (RF) Prediction:** RF Accuracy Score Using Default RF Params:
  - Runtime: 0.06175 seconds
  - Accuracy: 97.222%
- **Fastest Random Forest (RF) Prediction:** RF Runtime Using Default RF Params:
  - Runtime: 0.06175 seconds
  - Accuracy: 97.222%

The Random Forest (RF) model achieved the best accuracy of 97.22% with default settings, and the runtime was quite fast, at just 0.06175 seconds. This was also the fastest prediction for RF, as no bagging classifier was needed, and the default configuration produced this result in the same amount of time.

Results Cont.

- **Random Forest (RF) Confusion Matrix:**

The confusion matrix offers insight into the model's classification accuracy, showing the distribution of true positives, true negatives, false positives, and false negatives.



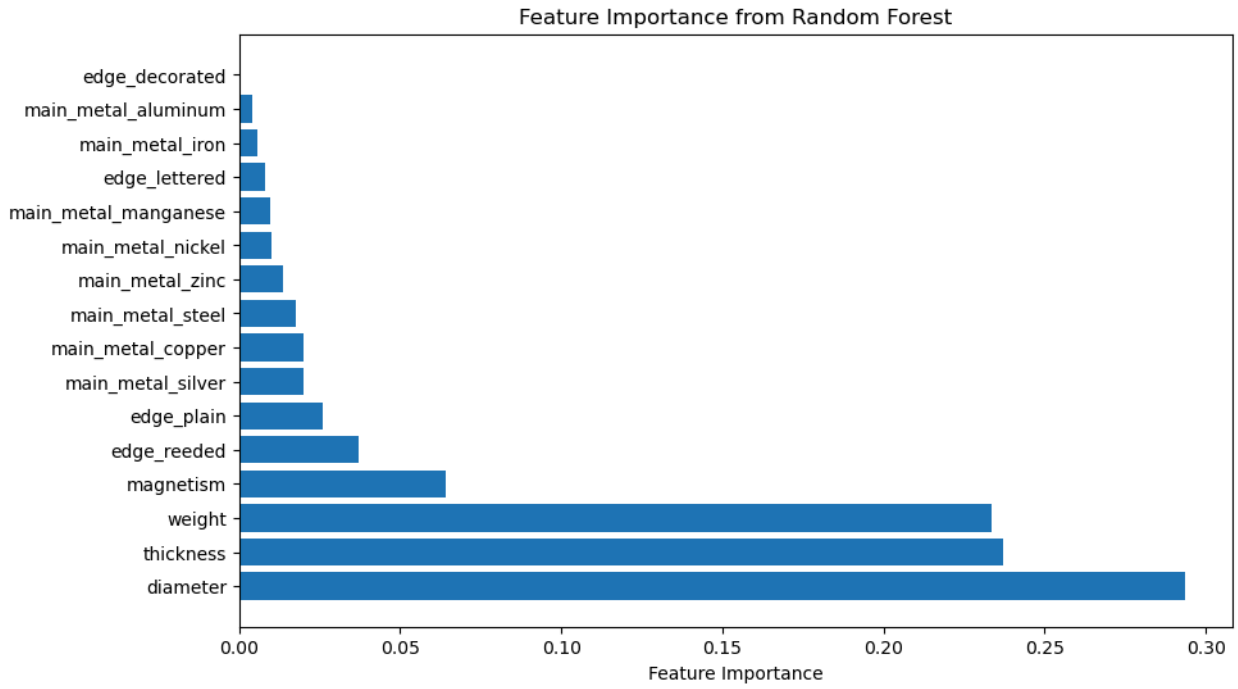
False Positive (FP): 3

False Negative (FN): 0

Results Cont.

- **Random Forest (RF) Feature Importance Graph:**

The graph below illustrates the relative importance of each feature in the Random Forest (RF) model.



Random Forest (RF) aggregates feature importance across multiple decision trees, providing a more robust measure of how each feature contributes to classification. The following percentages represent the average importance of each feature across all trees in the forest.

Results Cont.

- **Random Forest (RF) Feature Importance Comparison**

  1. **edge_decorated:** 0.0%
  2. **main_metal_aluminum:** 0.401%
  3. **main_metal_iron:** 0.585%
  4. **edge_lettered:** 0.801%
  5. **main_metal_manganese:** 0.977%
  6. **main_metal_nickel:** 1.007%
  7. **main_metal_zinc:** 1.36%
  8. **main_metal_steel:** 1.752%
  9. **main_metal_copper:** 1.981%
  10. **main_metal_silver:** 2.006%
  11. **edge_plain:** 2.6%
  12. **edge_reeded:** 3.727%
  13. **magnetism:** 6.41%
  14. **weight:** 23.358%
  15. **thickness:** 23.688%
  16. **diameter:** 29.346%

Results Cont.

The feature importance results from the Random Forest (RF) model indicate that physical dimensions played the most significant role in predicting whether a coin is legal U.S. tender. The diameter (29.346%), thickness (23.688%), and weight (23.358%) were the top three most influential features, showing that the overall size and mass of a coin are strong indicators in classification.

Among the remaining features, magnetism (6.41%) had the highest importance outside of physical measurements, suggesting that whether a coin is magnetic contributes meaningfully to the model's decision-making process. Edge characteristics, particularly edge_reeded (3.727%) and edge_plain (2.6%), also played a noticeable role, while edge_decorated (0.0%) had no influence on the model.

In terms of metal composition, no single metal type had a dominant impact, with main_metal_silver (2.006%), main_metal_copper (1.981%), and main_metal_steel (1.752%) contributing only marginally to the model's classification decisions. This suggests that while metal type has some influence, it is secondary to physical dimensions and magnetism.

Results Cont.

4. **Perceptron**

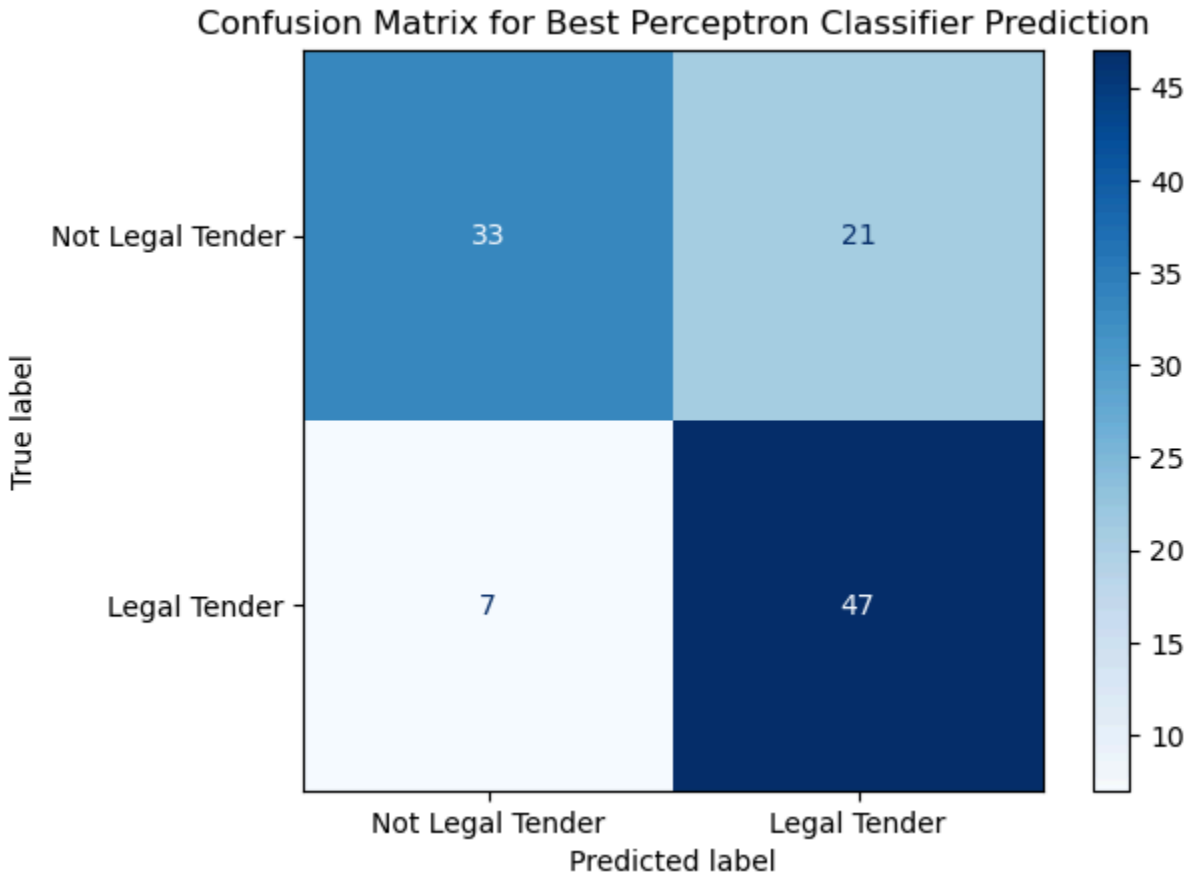
- **Highest Accuracy Perceptron Prediction:** Bag Accuracy Score Using Best Bag Params w/ Best Perceptron Params According To GridSearchCV:
  - Runtime: 55.96688 seconds
  - Accuracy: 74.074%
- **Fastest Perceptron Prediction:** Perceptron Runtime Using Default Perceptron Params:
  - Runtime: 0.00100 seconds
  - Accuracy: 72.222%

For the Perceptron model, the best accuracy was 74.07%, with the hyperparameters set as: alpha set to 0.001, early_stopping enabled, eta0 set to 1, max_iter set to 500, n_iter_no_change set to 25, penalty set to None, tol set to 1e-5, and validation_fraction set to 0.1. This configuration led to a runtime of 55.96688 seconds. The fastest prediction for Perceptron was achieved with the default parameters, completing in just 0.001 seconds but yielding a lower accuracy of 72.22%.

Results Cont.

- **Perceptron Confusion Matrix:**

The confusion matrix offers insight into the model's classification accuracy, showing the distribution of true positives, true negatives, false positives, and false negatives.



False Positive (FP): 21

False Negative (FN): 7

Results Cont.

5. **Logistic Regression (LR)**

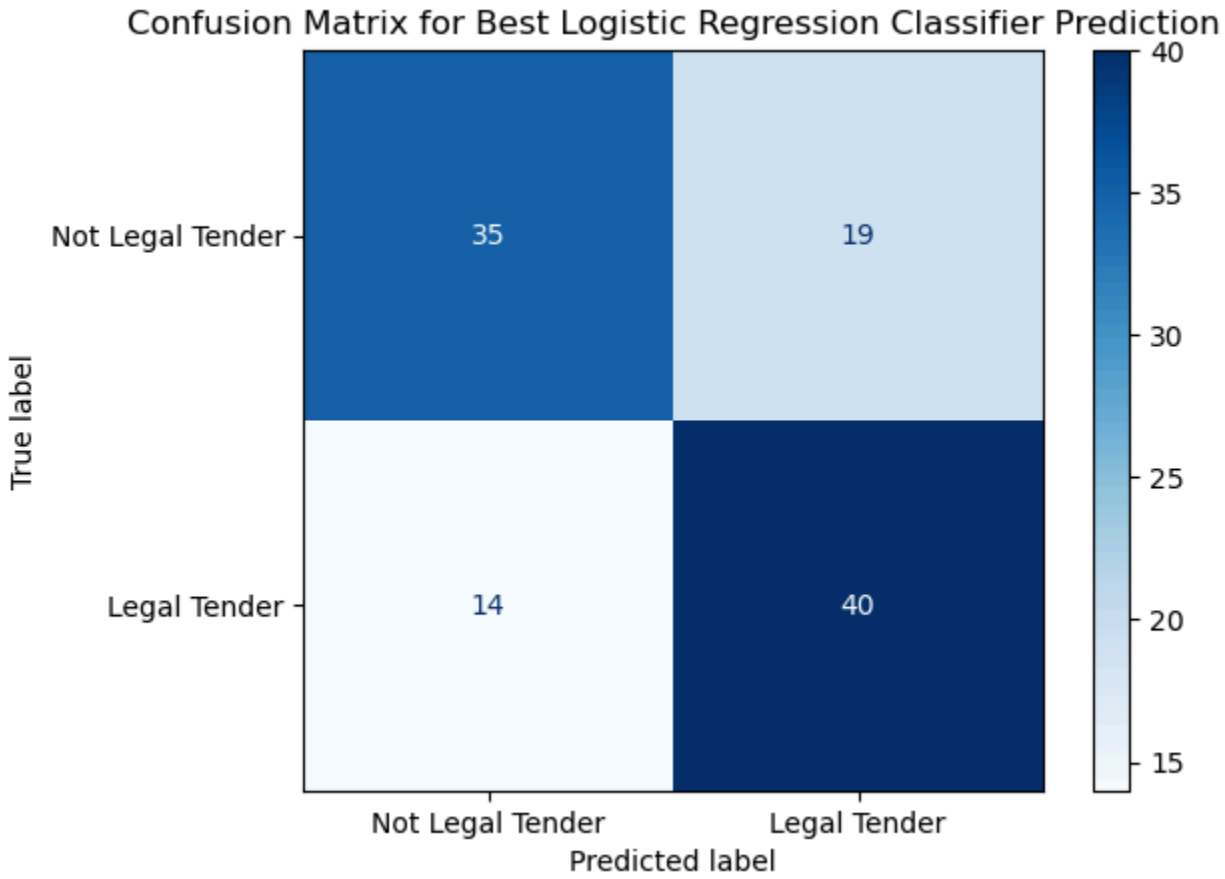
- **Highest Accuracy Logistic Regression (LR) Prediction:** Bag Accuracy Score Using Default Bag Params w/ Default LR Params:
  - Runtime: 1.84968 seconds
  - Accuracy: 69.444%
- **Fastest Logistic Regression (LR) Prediction:** LR Runtime Using Default LR Params:
  - Runtime: 0.00200 seconds
  - Accuracy: 67.593%

The Logistic Regression (LR) model reached a best accuracy of 69.44% with default settings. The runtime was 1.84968 seconds for the best configuration, and the fastest prediction completed in 0.002 seconds with the default parameters. Despite the quick predictions, the accuracy for LR remained on the lower end.

Results Cont.

- **Logistic Regression (LR) Confusion Matrix:**

The confusion matrix offers insight into the model's classification accuracy, showing the distribution of true positives, true negatives, false positives, and false negatives.



False Positive (FP): 19

False Negative (FN): 14

Conclusion

The goal of this project was to develop and evaluate machine learning models to predict whether a coin is legal U.S. tender based on various physical properties. I created a dataset containing 360 coins, where each entry included measurements such as diameter, thickness, weight, edge type, magnetism, and the main metal, along with the label indicating whether the coin was legal U.S. tender or not. The predictive models were then trained and evaluated to determine if these features could be effectively used to classify the coins.

Through this project, I applied appropriate data preprocessing techniques, such as MinMax scaling and label encoding, and tested multiple classifiers, including K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), Perceptron, and Logistic Regression (LR). These models were fine-tuned using GridSearchCV and RandomizedSearchCV, and bagging classifiers were incorporated to assess any performance gains.

The results revealed that the Decision Tree (DT) classifier provided the highest accuracy of 98.15%, closely followed by K-Nearest Neighbors (KNN) at 95.37%. The Random Forest (RF) and Perceptron models also showed reasonable accuracy, with Logistic Regression (LR) performing with the lowest accuracy but demonstrating quick prediction times. The comparison of the classifiers highlighted the balance between prediction accuracy and runtime, further emphasizing how different models and hyperparameter optimizations can impact performance in real-world tasks.

Conclusion Cont.

This project effectively demonstrates how machine learning can be applied to real-world classification tasks, such as predicting whether a coin is legal U.S. tender. It also shows how different models can be optimized and compared, highlighting the trade-offs between accuracy and prediction time. The project highlights the value of machine learning in classification problems and provides insight into the challenges and benefits of working with real-world datasets.

System Specifications

To ensure reproducibility, the experiment was conducted on the following machine:

- **Operating System:** Windows 11 Home (10.0.26100 Build 26100)
- **Processor:** AMD Ryzen 9 7900X (12 Cores, 24 Threads, 4.7 GHz)
- **Memory:** 64.0 GB RAM
- **Motherboard:** ASUS ROG STRIX B650E-F GAMING WIFI

Works Cited

- Numista, en.numista.com/. Accessed 16 Mar. 2025.
- NumPy, numpy.org/. Accessed 16 Mar. 2025.
- “Pandas.” Pandas, pandas.pydata.org/. Accessed 16 Mar. 2025.
- “Visualization with Python.” Matplotlib, matplotlib.org/. Accessed 16 Mar. 2025.
- “Statistical Data Visualization#.” Seaborn, seaborn.pydata.org/. Accessed 16 Mar. 2025.
- “Baggingclassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html. Accessed 16 Mar. 2025.
- “Decisiontreeclassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Accessed 16 Mar. 2025.
- “Kneighborsclassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed 16 Mar. 2025.
- “Logisticregression.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 16 Mar. 2025.
- “Perceptron.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html. Accessed 16 Mar. 2025.

Works Cited Cont.

- “Randomforestclassifier.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed 16 Mar. 2025.
- “Accuracy_score.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html. Accessed 16 Mar. 2025.
- “Confusionmatrixdisplay.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html. Accessed 16 Mar. 2025.
- “Confusion_matrix.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html. Accessed 16 Mar. 2025.
- “GRIDSEARCHCV.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed 16 Mar. 2025.
- “Labelencoder.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html. Accessed 16 Mar. 2025.
- “MinMaxScaler.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Accessed 16 Mar. 2025.

Works Cited Cont.

- “RANDOMIZEDSEARCHCV.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. Accessed 16 Mar. 2025.
- “Train_test_split.” Scikit, scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Accessed 16 Mar. 2025.
- “StandardScaler.” Scikit, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html Accessed 16 Mar.2025