**PANDIT DEENDAYAL ENERGY UNIVERSITY**

**SCHOOL OF TECHNOLOGY**

****

**Course: Artificial Intelligence Lab**

**Course Code: 23CP307**

**B.Tech. (Computer Engineering)**

**Semester 6**

|  |  |
| --- | --- |
| **Submitted To:**  Dr. Sidheswar Routray | **Submitted By:**  Heneel Parmar – 22BCP421  Rajdeep Dharajiya – 22BCP471D  Div-6, G11 |

**Title: Email Spam Classification Using Decision Tree Classifier a rule-based AI model.**

**Problem:** The goal of this project is to build a spam email classifier using a decision tree-based rule-based AI model. The system classifies emails as either spam (1) or non-spam (0). This task falls under supervised learning, with the primary objective being to predict whether an email is spam based on a set of input features.

**Dataset:** The dataset used in this project is the **SpamBase dataset** from the UCI Machine Learning Repository. The dataset contains 4,601 email instances, each described by 57 features and one target variable (label). The features represent various characteristics of the email, such as word frequencies and other statistics related to the content of the email. The target variable, label, indicates whether the email is spam (1) or non-spam (0).

**Key Features:**

* feature1 to feature57: These are various features that represent characteristics of the email (e.g., frequency of specific words, number of capital letters, etc.).
* label: The target variable (1 for spam, 0 for non-spam).

**Methodology:**

1. **Data Preprocessing:**

* **Missing Values Handling**: No missing values were found in the dataset.
* **Duplicate Removal**: 391 duplicate entries were removed.
* **Data Splitting**: The data was split into training (80%) and testing (20%) sets.

2. **Model Development**:

* A **Decision Tree Classifier** was used as the AI model. This is a rule-based machine learning model that works by splitting the dataset into subsets based on feature values, thereby creating decision rules for classification.

3. **Evaluation Metrics**:

**Evaluation Metrics Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Class 0 (Not Spam)** | **Class 1 (Spam)** | **Overall / Weighted Avg** |
| **Precision** | 0.92 | 0.92 | 0.92 |
| **Recall** | 0.94 | 0.88 | 0.92 |
| **F1-Score** | 0.93 | 0.90 | 0.92 |
| **Support** | 531 | 390 | 921 |
| **Accuracy** |  |  | **91.86%** |

**Confusion Matrix:**

|  |  |  |
| --- | --- | --- |
|  | **Predicted: Not Spam (0)** | **Predicted: Spam (1)** |
| **Actual: Not Spam (0)** | 501 (True Negative) | 30 (False Positive) |
| **Actual: Spam (1)** | 45 (False Negative) | 345 (True Positive) |

* Accuracy, Precision, Recall, F1-score, and Confusion Matrix were used to evaluate the model's performance.

1**. Precision**

* **Precision for class 1 (Spam):** 0.92  
  → Out of all emails predicted as spam, **92% were actually spam**.
* **Why it matters:** Helps avoid **false positives** (e.g., marking important emails as spam).

**2. Recall**

* **Recall for class 1 (Spam):** 0.88  
  → Out of all real spam emails, **88% were correctly detected** as spam.
* **Why it matters:** Shows how well your model catches spam — important to reduce **missed spam**.

**3. F1-Score**

* **F1-score (for class 1):** 0.90  
  → Harmonic mean of precision and recall — balances both.
* **Why it matters:** A good single score that balances both **false positives** and **false negatives**.

**4. Accuracy**

* Overall: **91.86%**  
  → Out of 921 total emails, 846 were classified **correctly** (501 TN + 345 TP).
* **Why it matters:** It’s the simplest metric, but can be misleading if classes are imbalanced — which is why **precision/recall** are also used.

**5. Confusion Matrix (detailed):**

* **True Positives (TP = 345):** Spam emails correctly marked as spam.
* **True Negatives (TN = 501):** Non-spam emails correctly marked as not spam.
* **False Positives (FP = 30):** Non-spam emails **wrongly marked as spam**.
* **False Negatives (FN = 45):** Spam emails that were **missed** by the model.

**4. Cross-validation**:

* Cross-validation with 10 folds was performed to evaluate the model's stability across different subsets of the data.

**Results:**

* The final accuracy of the model was **91.86%**, with a confusion matrix showing **501 true negatives**, **345 true positives**, **30 false positives**, and **45 false negatives**.
* The classification report showed that the model performed well in terms of precision, recall, and F1-score, with an overall weighted average F1-score of 0.92.
* The average cross-validation score was **0.90**, indicating consistent performance across different data splits.

**Challenges:**

* **Overfitting**: The model showed good performance on both the training and test datasets, but further exploration of hyperparameters could help improve the generalization.
* **Feature Selection**: The dataset contains many features, some of which may not contribute significantly to the classification task. Feature selection or dimensionality reduction could be explored to improve model efficiency.

**AI Approach:**

The **Decision Tree Classifier** was chosen as the model due to its interpretability and rule-based structure. Decision trees are easy to understand and can be visualized, making them ideal for rule-based AI systems.

**Solution:**

* **Data Preprocessing**: Handled missing values and duplicates.
* **Model Training**: The Decision Tree classifier was trained using the preprocessed dataset, with an accuracy of 91.86% on the test data.
* **Evaluation**: Evaluated using precision, recall, F1-score, and confusion matrix.

**Results:**

* **Accuracy**: 91.86%
* **Precision**: 0.92 for both classes
* **Recall**: 0.94 for class 0 and 0.88 for class 1
* **F1-Score**: 0.93 for class 0 and 0.90 for class 1
* **Confusion Matrix**:
  + True Negatives: 501
  + True Positives: 345
  + False Positives: 30
  + False Negatives: 45

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import cross\_val\_score

# 1. Load the dataset

# Replace the file path with the location where you've saved the dataset

file\_path = 'spambase.data'  # Modify this path accordingly

columns = [f'feature{i}' for i in range(1, 58)] + ['label']  # Columns for features and target variable

df = pd.read\_csv(file\_path, header=None, names=columns)

# 2. Data Preprocessing

# Handling missing values (if any)

print("Missing values:\n", df.isnull().sum())

# Check for duplicates

print("\nDuplicates in the dataset:", df.duplicated().sum())

# Check the first few records

print("\nData Preview:")

print(df.head())

# Feature and target separation

X = df.drop('label', axis=1)  # Features

y = df['label']  # Target (label)

# 3. Data Visualization

# Visualizing feature correlations

plt.figure(figsize=(12, 8))

sns.heatmap(X.corr(), annot=False, cmap='coolwarm', fmt='.1f', linewidths=0.5)

plt.title("Feature Correlations")

plt.show()

# 4. Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 5. Decision Tree Classifier (Rule-based AI)

# Initialize the Decision Tree classifier

clf = DecisionTreeClassifier(random\_state=42)

# Train the model

clf.fit(X\_train, y\_train)

# 6. Model Prediction

y\_pred = clf.predict(X\_test)

# 7. Model Evaluation

# Accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

# Confusion Matrix

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

# Classification Report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# 8. Handling Overfitting & Underfitting

# Cross-validation to check for overfitting/underfitting

cross\_val\_scores = cross\_val\_score(clf, X, y, cv=10)

print(f'\nCross-validation scores: {cross\_val\_scores}')

print(f'Average Cross-validation score: {cross\_val\_scores.mean()}')

# 9. Feature Importance (Optional, to see which features are most important)

feature\_importances = clf.feature\_importances\_

indices = np.argsort(feature\_importances)[::-1]

plt.figure(figsize=(12, 6))

plt.title("Feature Importances")

plt.barh(range(X.shape[1]), feature\_importances[indices], align='center')

plt.yticks(range(X.shape[1]), [f'feature{i+1}' for i in indices])

plt.xlabel('Relative Importance')

plt.show()

# 10. Handling Overfitting with Pruning (Optional)

# You can prune the decision tree using parameters like max\_depth to avoid overfitting

clf\_pruned = DecisionTreeClassifier(random\_state=42, max\_depth=5)

clf\_pruned.fit(X\_train, y\_train)

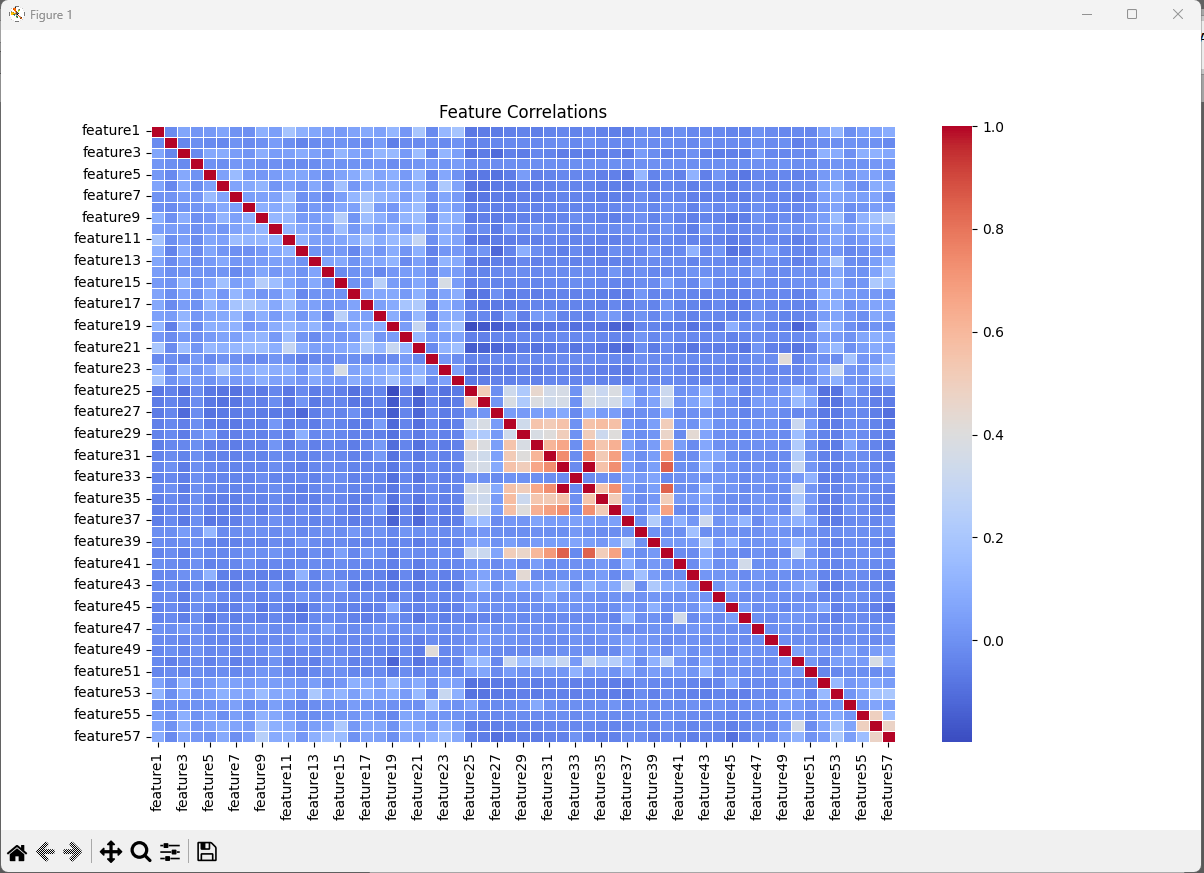
y\_pred\_pruned = clf\_pruned.predict(X\_test)

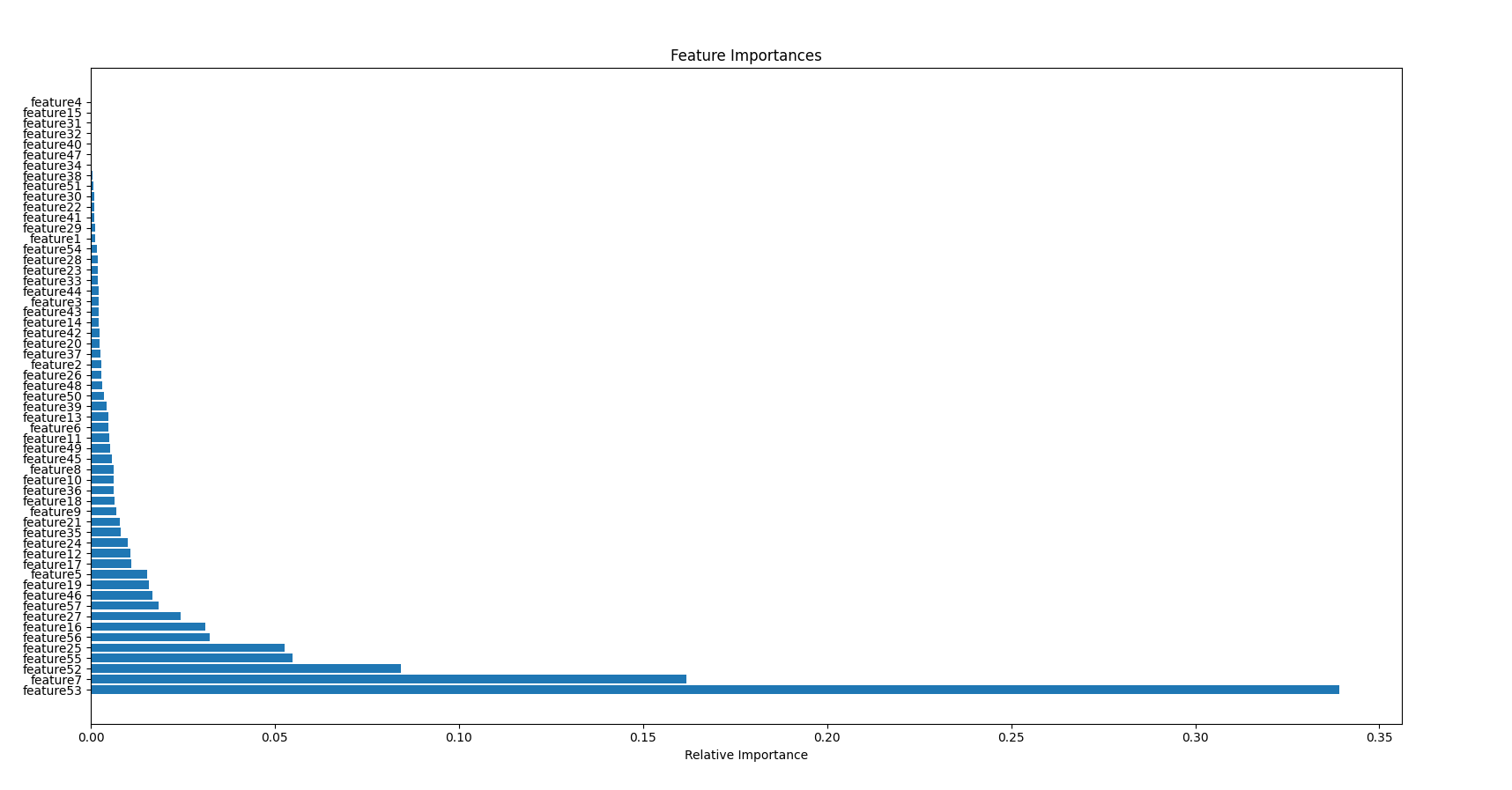
# Pruned Model Evaluation

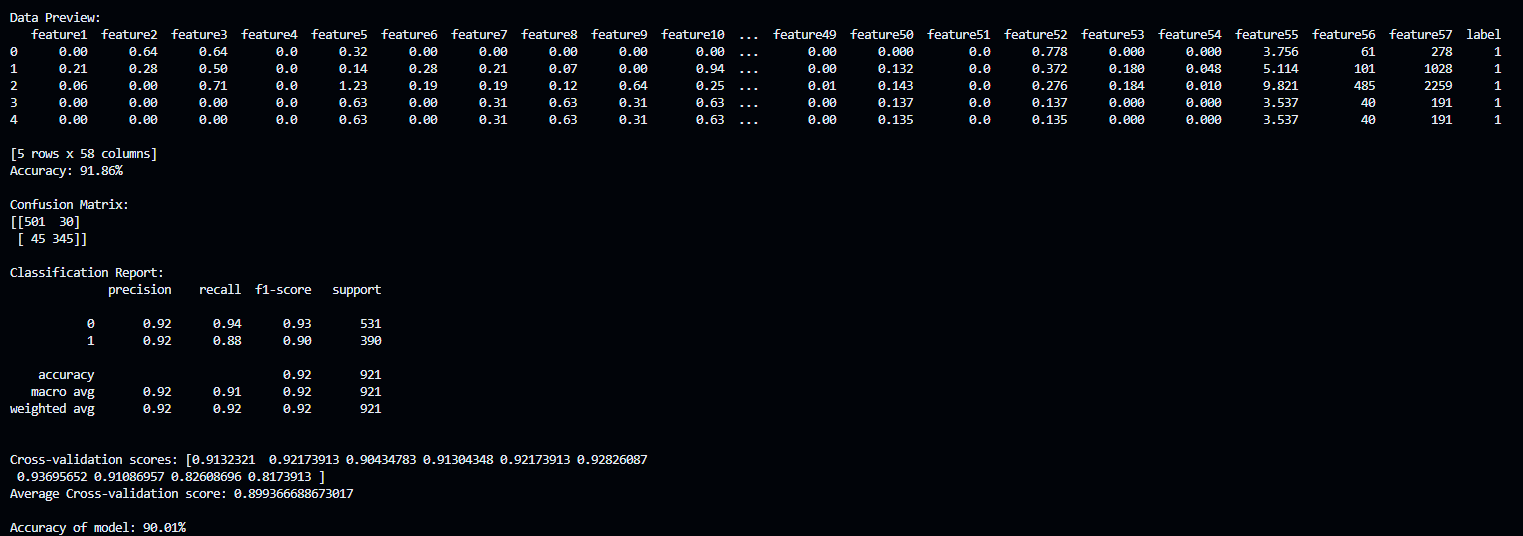
accuracy\_pruned = accuracy\_score(y\_test, y\_pred\_pruned)

print(f'\nAccuracy of model: {accuracy\_pruned \* 100:.2f}%')

**Output:**







**Conclusion:**

This project successfully developed an AI-driven system for email spam classification using a Decision Tree Classifier. The model was trained on the SpamBase dataset, achieving a test accuracy of 91.86%. Data preprocessing steps, including duplicate removal, were crucial in preparing the data for training. While the model performed well, future work could explore feature selection and hyperparameter optimization to further enhance its generalization and efficiency.