DISTINGUISHING

REAL VS AI-GENERATED IMAGE TEXTURES

PRESENTED BY

TEAM A

PROBLEM STATEMENT

This paper aims to compare the efficacy of machine learning models (e.g., SVM, KNN, Naïve Bayes, Random Forest, Logistic Regression) on texture-based features from actual and generated images, and identify the highest-performing classifier.

INTRODUCTION

AI-generated image synthesis with GANs and diffusion models has transformed digital media but also threatens to cause risks such as misinformation and identity theft. Texture-based analysis provides a strong method for identifying AI-generated images by detecting subtle structural patterns finer than pixel intensity. GLCM, LBP, and wavelet transforms, as well as deep learning embeddings, improve detection accuracy. To facilitate research in this area, we present the Real vs AI-Generated Image Texture Feature Dataset, consisting of pre-extracted texture features for training and testing machine and deep learning models to detect AI-generated content, with applications in AI ethics, cybersecurity, and forensic analysis.

LITERATURE REVIEW

Feature Used	ML/AI Models Used	Study Results	Source
Anatomical Implausibilities (e.g., distorted hands, eyes)	Human visual analysis, pattern recognition	Effective in identifying many, but not all, AI-generated images. Relies on human observation.	https://insight.kellogg.northwestern.edu/article/ai-photos-identification
Metadata Analysis	Metadata extraction tools	Shows state of the art results, by measuring the uncertainty of the model when classifying images.	https://arxiv.org/html/2412.05897v1
Frequency Domain Analysis	Fourier transforms, spectral analysis, deep learning classifiers	Reveals subtle differences in image structure, improving detection accuracy.	https://www.mdpi.com/2078-2489/15/11/711
Machine Learning Classifiers (trained on real vs. AI-generated datasets)	Convolutional Neural Networks (CNNs), Vision Transformer (ViT)	Achieves high accuracy in controlled environments, but performance can vary with new AI-generated image types.	https://openreview.net/forum?id=oClr2P7V0T

METHODOLOGY

1. DATASET OVERVIEW

Source: Kaggle Dataset: Real vs AI-Generated Image Texture Feature Dataset.

Description: The dataset includes GLCM characteristics and Shannon entropy computed for real and AI-generated images.

Data Characteristics: The dataset contains 66898 rows and 123 columns, with 122 'float' features and a 'label' column.

Data Distribution: The dataset has 33477 actual images and 33421 AI-generated images.

2. MATHEMATICAL MODELS

A. DATA PREPROCESSING

- Removal of unnecessary features (serial numbers):

```
df.drop(columns=['Unnamed: 0'], inplace=True) # dropping the unnecessary column
```

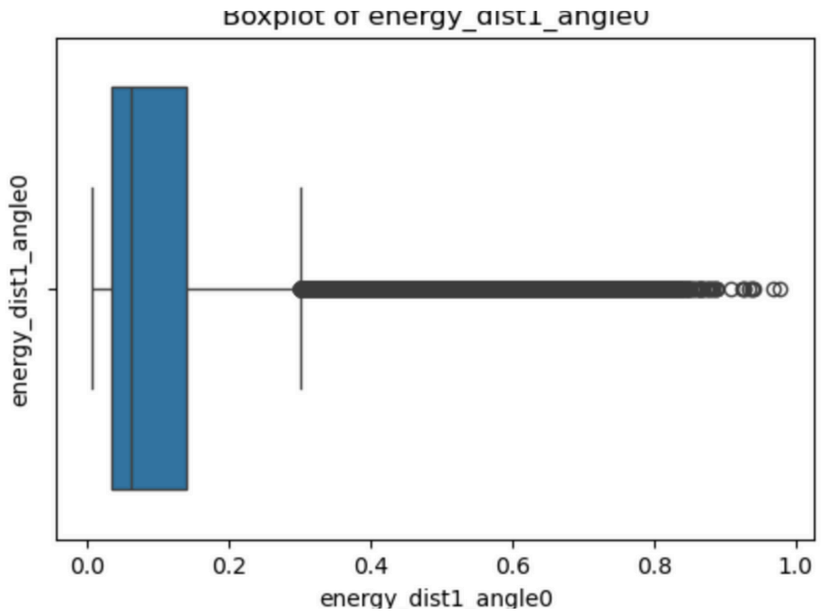
- Handling missing values with Median:

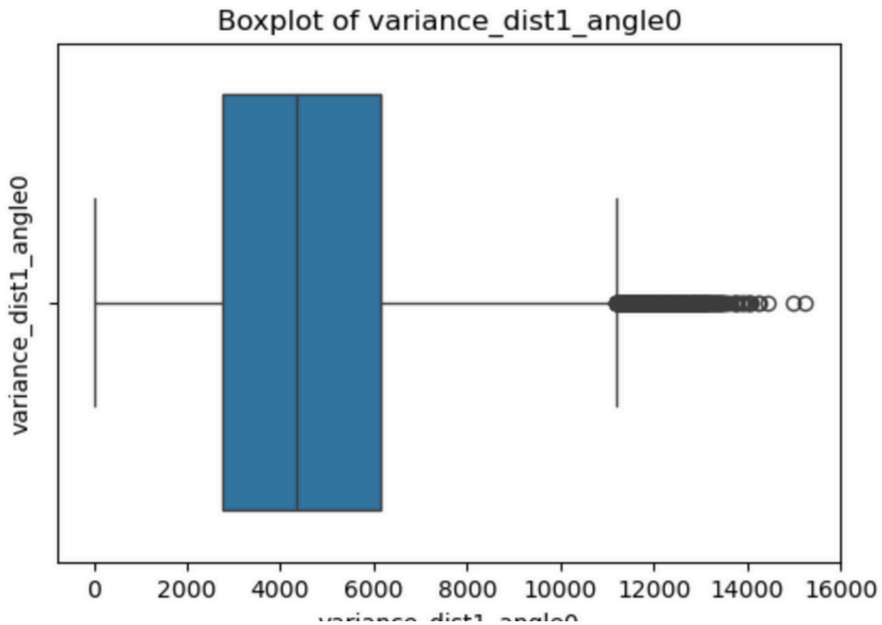
```
df.isnull().sum() # checking for null values
```

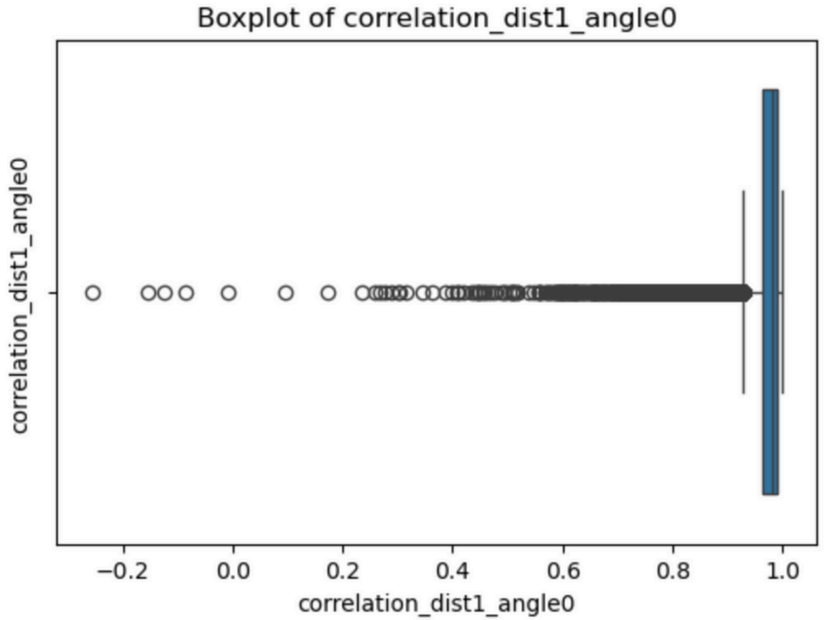
```
contrast_dist1_angle0         0
dissimilarity_dist1_angle0    0
homogeneity_dist1_angle0      0
energy_dist1_angle0           0
correlation_dist1_angle0      0
                             ..
variance_dist5_angle135       1
std_dist5_angle135            1
entropy_dist5_angle135        1
shannon_entropy               1
labels                        1
Length: 122, dtype: int64
```

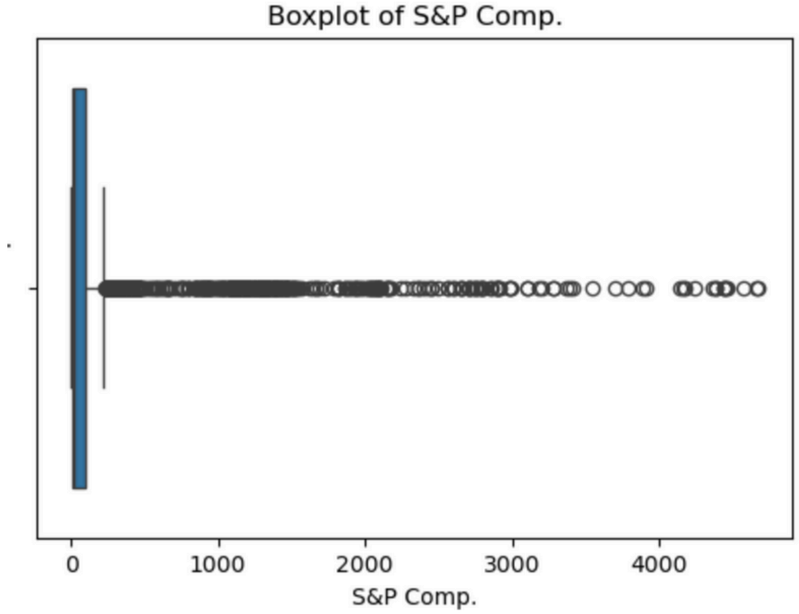
```
df.fillna(df.median(), inplace=True) # filling missing values with median
```

- Outliers in the Dataset (Detection uisng Boxplots)









- We use IQR in the Capping Method for removal of outliers.

```python
# Capping using IQR

def cap_outliers_iqr(df):
    for col in df.select_dtypes(include=['float64']).columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
        df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])

    return df
```

```python
df = cap_outliers_iqr(df)
df.head(2)
```

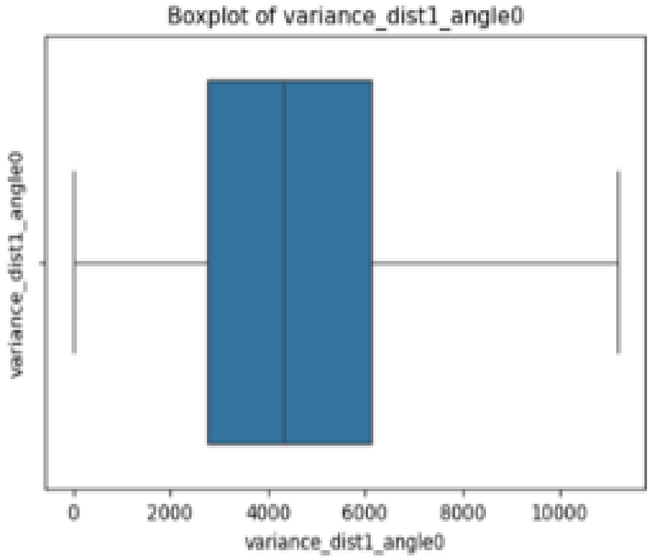
$$IQR = Q_3 - Q_1$$

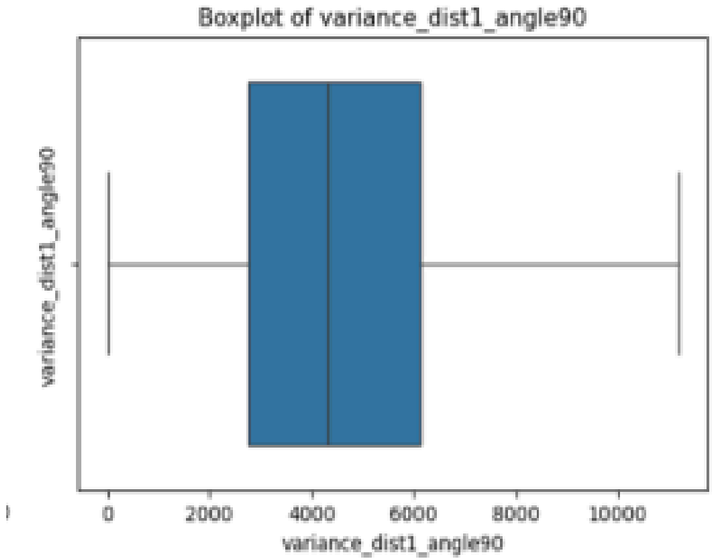
$$\text{Lower Bound} = Q_1 - 1.5 \times IQR$$

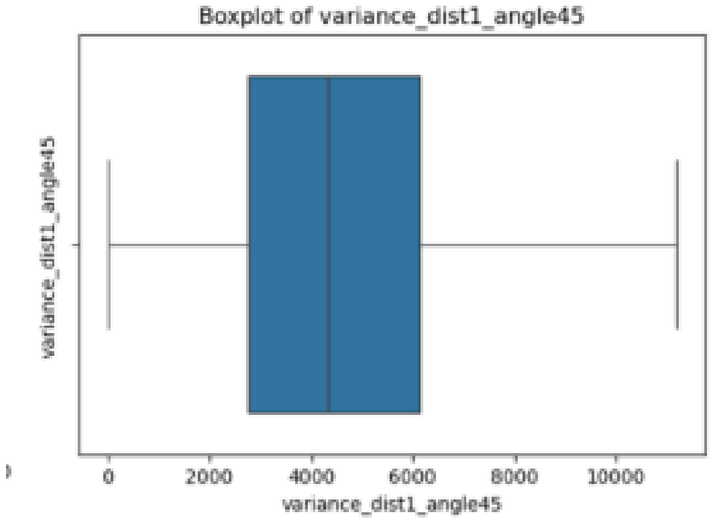
$$\text{Upper Bound} = Q_3 + 1.5 \times IQR$$

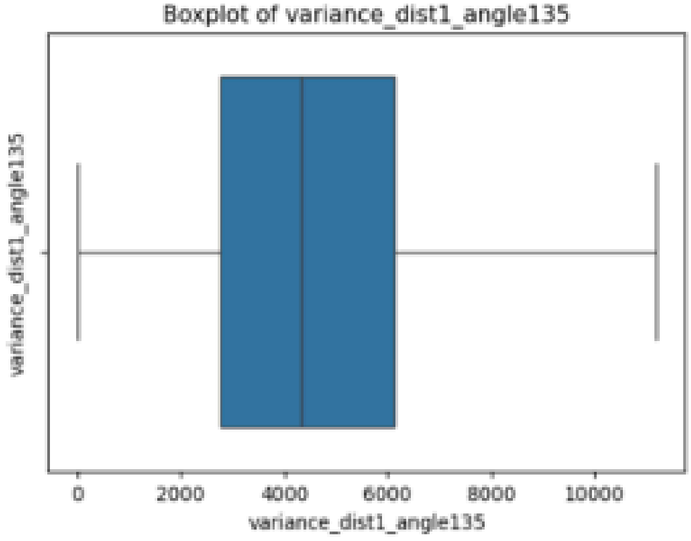
IQR (Interquartile Range) is a measure of statistical dispersion, representing the middle 50% of a dataset.

Boxplots after the Capping of Outliers









STANDARDIZATION OF DATA:

Here, we use StandardScaler() to standardize our data

```
from sklearn.preprocessing import StandardScaler

# Standardization features
scaler = StandardScaler()
X_sampled_scaled = scaler.fit_transform(X_sampled)
```

The mathematical formula for **data standardization (Z-score normalization)** is:

$$Z = \frac{X - \mu}{\sigma}$$

Where:

- Z = Standardized value
- X = Original value
- μ = Mean of the feature
- σ = Standard deviation of the feature

This transformation ensures that the standardized data has a **mean of 0** and a **standard deviation of** 1

DIMENSIONALITY REDUCTION BY FEATURE SELECTION

(PEARSON'S COEFFICIENT CORRELATION):

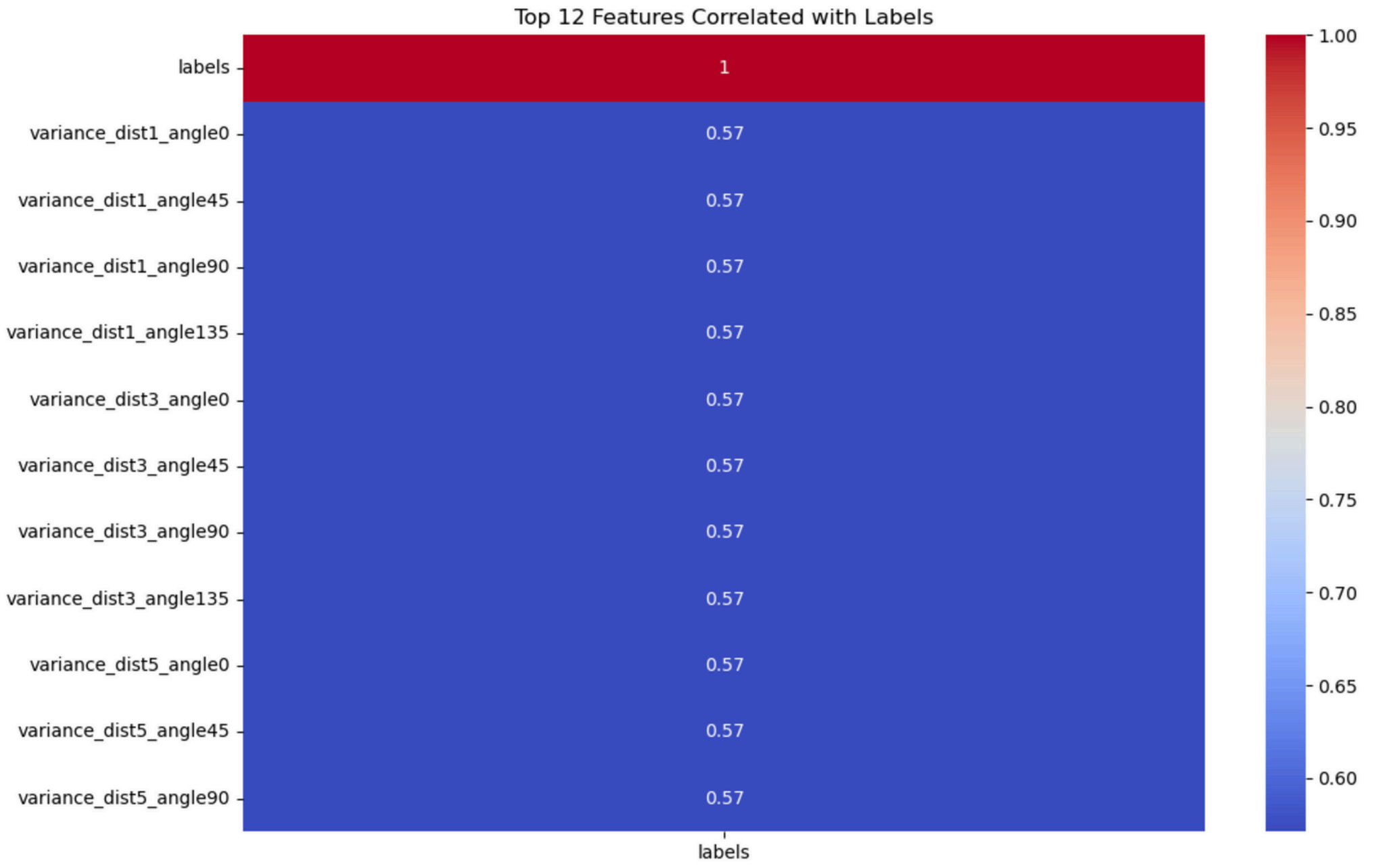
- Only the top 12 features are selected.
- Visualised by Correlation Heatmap

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2} \cdot \sqrt{\sum(Y_i - \bar{Y})^2}}$$

Where:

- X_i, Y_i = Individual data points of variables X and Y
- $\bar{X}, \bar{Y}$ = Mean of X and Y
- $\sum$ = Summation across all data points

```
# Correlation heatmap (only top features to avoid clutter)
plt.figure(figsize=(12,8))
correlation_matrix = df.corr().abs()
sns.heatmap(correlation_matrix.nlargest(12, 'labels')['labels'].to_frame(), annot=True, cmap='coolwarm')
plt.title('Top 12 Features Correlated with Labels')
plt.show()
```


Top 12 Features Correlated with Labels
labels
variance_dist1_angle0
variance_dist1_angle45
variance_dist1_angle90
variance_dist1_angle135
variance_dist3_angle0
variance_dist3_angle45
variance_dist3_angle90
variance_dist3_angle135
variance_dist5_angle0
variance_dist5_angle45
variance_dist5_angle90
1
0.57
labels
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60

FEATURE EXTRACTION USING PRINCIPAL COMPONENT ANALYSIS (PCA) :

- Used to reduce the dimensions
- Using PCA top 4 PC's are selected on scaled data.

Principle Component Analysis

```
from sklearn.decomposition import PCA

# Applying PCA with 4 components
pca_sampled = PCA(n_components=4)
X_sampled_pca = pca_sampled.fit_transform(X_sampled_scaled)
```

```
df_pca = pd.DataFrame(X_sampled_pca, columns=['PC1','PC2','PC3','PC4'])
df_pca.head()
```

	PC1	PC2	PC3	PC4
0	-5.573318	0.594632	5.376433	2.065969
1	8.300547	-1.329511	-0.125407	-2.417016
2	2.686605	-3.407956	-0.159570	2.956864
3	1.901859	-1.088109	-3.222946	4.634423
4	-7.541557	-0.679947	0.742298	2.082348

```
explained_variance = pca_sampled.explained_variance_ratio_
print(f"Explained Variance: {explained_variance}")
print(f"Total Variance Captured: {sum(explained_variance)*100:.2f}%")
```

```
Explained Variance: [0.48912086 0.23685052 0.15936595 0.06820974]
Total Variance Captured: 95.35%
```

MODEL SELECTION:

The ML models used are :

1. Logistic Regression,
2. Random Forest Classification
3. Naïve Bayes Model
4. KNN and
5. SVM with linear, polynomial, and RBF kernels.

Model Fitting

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Initializing models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Naïve Bayes": GaussianNB(),
    "KNN" : KNeighborsClassifier(n_neighbors=5),
    "SVM - linear kernel": SVC(kernel="linear", random_state=42),
    "SVM - polynomial kernel": SVC(kernel="poly", random_state=42),
    "SVM - RBF kernel": SVC(kernel="rbf", random_state=42)
}
```

B. MACHINE LEARNING ALGORITHMS

LOGISTIC REGRESSION:

The simple mathematical formula for **Logistic Regression** is:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(wX+b)}}$$

where:

- $P(Y = 1|X)$ is the probability of class **1**
- w is the weight vector
- X is the feature vector
- b is the bias term
- e is Euler's number (≈ 2.718)

For classification:

$$\hat{y} = \begin{cases} 1, & \text{if } P(Y = 1|X) \geq 0.5 \\ 0, & \text{if } P(Y = 1|X) < 0.5 \end{cases}$$

This formula ensures that the output is between 0 and 1, making it useful for **binary classification**.

Mathematical Formulation

```
Algorithm: Logistic Regression

Input:
    - Training data (X_train, y_train)
    - Learning rate (α)
    - Number of iterations (epochs)

Step 1: Initialize Weights
    - Set weights (w) and bias (b) to small random values

Step 2: Repeat until convergence (for specified epochs):
    - Compute linear combination: z = X * w + b
    - Apply sigmoid function: y_pred = 1 / (1 + exp(-z))
    - Compute loss using binary cross-entropy
    - Compute gradients:
        - dw = (1/m) * Σ (y_pred - y) * X
        - db = (1/m) * Σ (y_pred - y)
    - Update weights:
        - w = w - α * dw
        - b = b - α * db

Step 3: Prediction
    - For new input X_new, compute z = X_new * w + b
    - Apply sigmoid function and classify:
        - If sigmoid(z) ≥ 0.5, predict 1
        - Else, predict 0

Output:
    - Optimized weights (w, b)
    - Predicted labels for test data
```

Pseudocode

RANDOM FOREST CLASSIFIER:

For **classification**, Random Forest predicts the most common class (mode) among all decision trees:

$$\hat{y} = \text{mode}\{h_1(X), h_2(X), ..., h_T(X)\}$$

For **regression**, it takes the average of all decision tree outputs:

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T} h_t(X)$$

where:

- T is the total number of decision trees
- $h_t(X)$ is the prediction from tree t
- **mode** is the most frequently occurring class (for classification)
- **mean** is used for regression predictions

Mathematical Formulation

Pseudocode

```
Algorithm: Random Forest

Input: Training dataset (X, y), number of trees (T)
Output: Trained Random Forest model

1. For t = 1 to T:
    a. Sample a random subset of training data with replacement (Bootstrap sampling
    b. Train a Decision Tree on the sampled data
    c. At each node of the tree:
       - Select a random subset of features
       - Split using the best feature based on a chosen criterion (e.g., Gini Impur

2. Store all trained trees

3. For a new input X:
    a. Get predictions from each tree
    b. Use majority voting (for classification) or averaging (for regression) to de

4. Return the final prediction
```

NAÏVE BAYES MODEL

Bayes' Theorem:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Where:

- $P(C|X)$ = Probability of class C given features X (posterior probability)
- $P(X|C)$ = Probability of features X given class C (likelihood)
- $P(C)$ = Probability of class C occurring (prior)
- $P(X)$ = Probability of features X occurring (evidence)

Mathematical Formulation

```
Algorithm: Naïve Bayes Classifier

Input: Training data (features X, labels y), Test data (features X_test)

Step 1: Train the Model
- Compute prior probability for each class:
  P(Class) = (Count of samples in Class) / (Total samples)
- Compute likelihood for each feature:
  P(Feature | Class) = (Count of Feature in Class + α) / (Total samples in Class + α

Step 2: Predict Class for New Data
- For each test sample, compute posterior probability for each class:
  P(Class | X) = P(Feature1 | Class) * P(Feature2 | Class) * ... * P(Class)
- Assign the class with the highest probability

Step 3: Evaluate Model
- Compare predicted labels with actual labels
- Calculate accuracy, precision, recall, and F1-score

Output: Predicted class labels for test data, Model performance metrics
```

Pseudocode

K-NEAREST NEIGHBOURS (KNN)

The most commonly used distance metric in KNN is **Euclidean Distance**, given by:

$$d(i,j) = \sqrt{\sum_{n=1}^{N} (X_{i,n} - X_{j,n})^2}$$

Where:

- $d(i,j)$ = Distance between points i and j
- $X_{i,n}$ = Feature n of data point i
- $X_{j,n}$ = Feature n of data point j
- N = Number of features

Mathematical Formulation

```
Algorithm: K-Nearest Neighbors (KNN)
Input: Training dataset (X_train, y_train), Test dataset (X_test), Number of neighbor
Output: Predicted labels for X_test

1. For each test instance x in X_test:
   a. Compute the distance between x and all instances in X_train
   b. Sort the distances in ascending order
   c. Select the k-nearest neighbors (smallest k distances)
   d. For classification:
      - Count the occurrences of each class in k-nearest neighbors
      - Assign the most frequent class as the predicted label for x
   e. For regression:
      - Compute the average value of the k-nearest neighbors
      - Assign this average as the predicted value for x

2. Return the predicted labels for all test instances.
```

Pseudocode

SUPPORT VECTOR MACHINE :

(i) Linear Kernel :

- **Formula:**

$$f(X) = \text{sign}(w^T X + b)$$

where w is the weight vector and b is the bias.

- **Pseudocode:**

```mathematica
Initialize w, b
Repeat until convergence:
    For each sample (X_i, y_i):
        If sample is misclassified:
            Update w and b
Return w and b
```

(ii) Polynomial Kernel SVM:

- **Formula (Polynomial Kernel):**

$$K(X_i, X_j) = (X_i^T X_j + c)^d$$

where d is the degree and c is a constant.

- **Pseudocode:**

```r

Initialize α, b
Compute kernel K(X_i, X_j) = (X_i^T X_j + c)^d
Train model using kernel trick
Return decision function
```

(iii) RBF (Radial Basis Function) Kernel SVM:

3. SVM with RBF Kernel

- **Formula (RBF Kernel):**

$$K(X_i, X_j) = \exp(-\gamma ||X_i - X_j||^2)$$

  where γ controls the spread.

- **Pseudocode:**

```sql
Initialize α, b
Compute kernel K(X_i, X_j) = exp(-γ ||X_i - X_j||^2)
Train model using kernel trick
Return decision function
```

Each SVM type uses a different kernel to transform data for better separation.

C. PERFORMANCE METRICS

CONFUSION MATRIX:

- **Key Terms**:
- True Positives (TP) – Model correctly predicts a real image as real.
- True Negatives (TN) – Model correctly predicts an AI-generated image as AI-generated.
- False Positives (FP) – Model incorrectly classifies an AI-generated image as real (Type I Error).
- False Negatives (FN) – Model incorrectly classifies a real image as AI-generated (Type II Error).

Performance Metrics derived from Confusion Matrix:

Evaluation Metrics:

- **Accuracy:** Measures correct predictions.
- **Precision & Recall:** Important for imbalanced datasets.
- **F1-score**: Balances precision and recall.

Confusion Matrix:

Confusion matrices are used to visualize the model's performance.

Confusion Matrix

		Actual	
		True	**False**
Predicted	**True**	True Positive (TP)	False Positive (FP) (Type I error)
	False	False Negative (FN) (Type II error)	True Negative (TN)

The performance metrics that we obtain from sklearn.metrics library are given below:

Accuracy (Overall Correctness): shows the proportion of correctly classified instances by each model.

Precision (Correct Positive Predictions): indicates how often the model was correct when it predicted a positive outcome.

Recall (Found All Positive Cases): shows the model's ability to find all the actual positive cases.

F1-Score (Balance): provides a balance between precision and recall.

Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision (Positive Predictive Value):

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity or True Positive Rate):

$$Recall = \frac{TP}{TP + FN}$$

F1-Score:

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

3. PROCEDURE

Case Study 1: Feature Selection (Filter Method)

Here, we chose the best 12 features from 122 by the Filter Method of Pearson's Correlation Coefficient with the target variable. The 12 features explained 91.03% of the variation in the data set and are a good representative of the original data. We created a reduced-dimensional data frame and trained Machine Learning models on this improved dataset to test their accuracy.

Case Study 2: Feature Extraction (PCA)

Applying Principal Component Analysis (PCA), we collapsed the 122 features into 4 Principal Components (PCs) that accounted for 95.35% of the cumulative variance. We then trained Machine Learning models using these components and compared their performances.

Comparative analysis was done to identify how these methods of dimension reduction affected model accuracy, training effectiveness, and predictability.

RESULTS
&
DISCUSSION

Performance Metrics:

- This includes the Accuracy score and classification report

Accuracy Scores of Feature Selection (Filter)

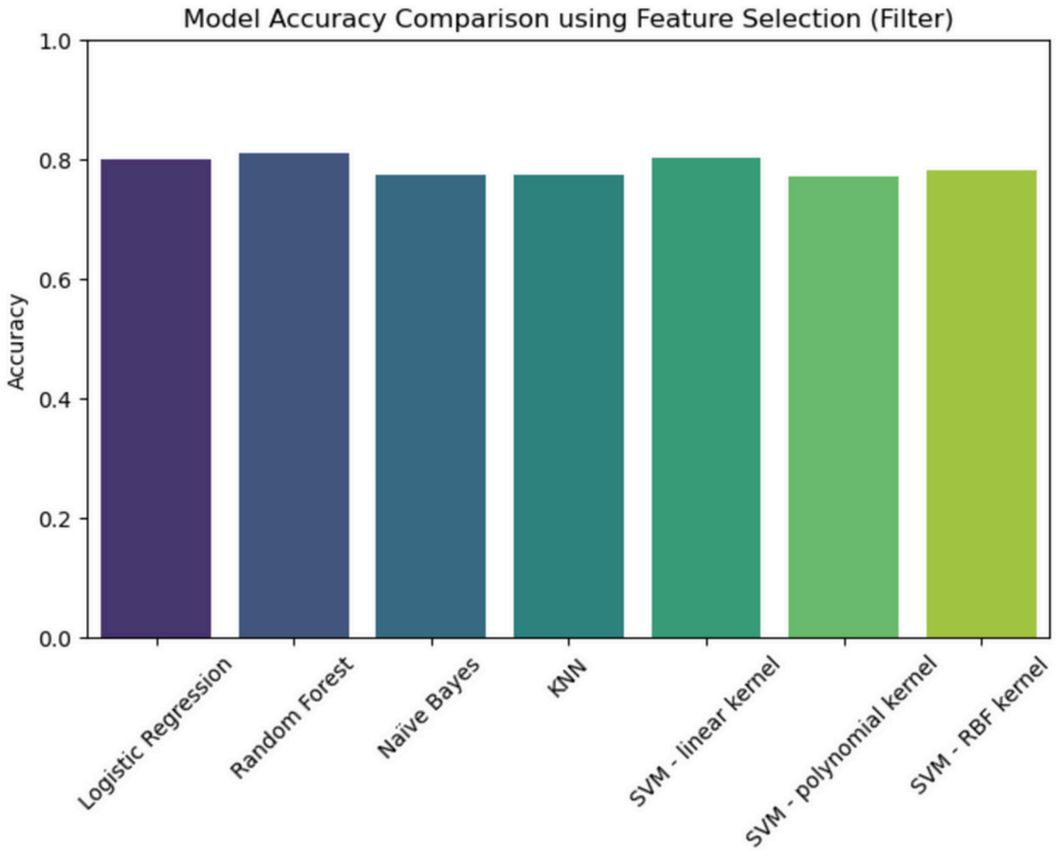
	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.7985	0.800221	0.7985	0.798463
Random Forest	0.8100	0.810165	0.8100	0.810030
Naïve Bayes	0.7745	0.775279	0.7745	0.774532
KNN	0.7740	0.774171	0.7740	0.774036
SVM - linear kernel	0.8020	0.803178	0.8020	0.801542
SVM - polynomial kernel	0.7700	0.781734	0.7700	0.768453
SVM - RBF kernel	0.7825	0.784649	0.7825	0.782412

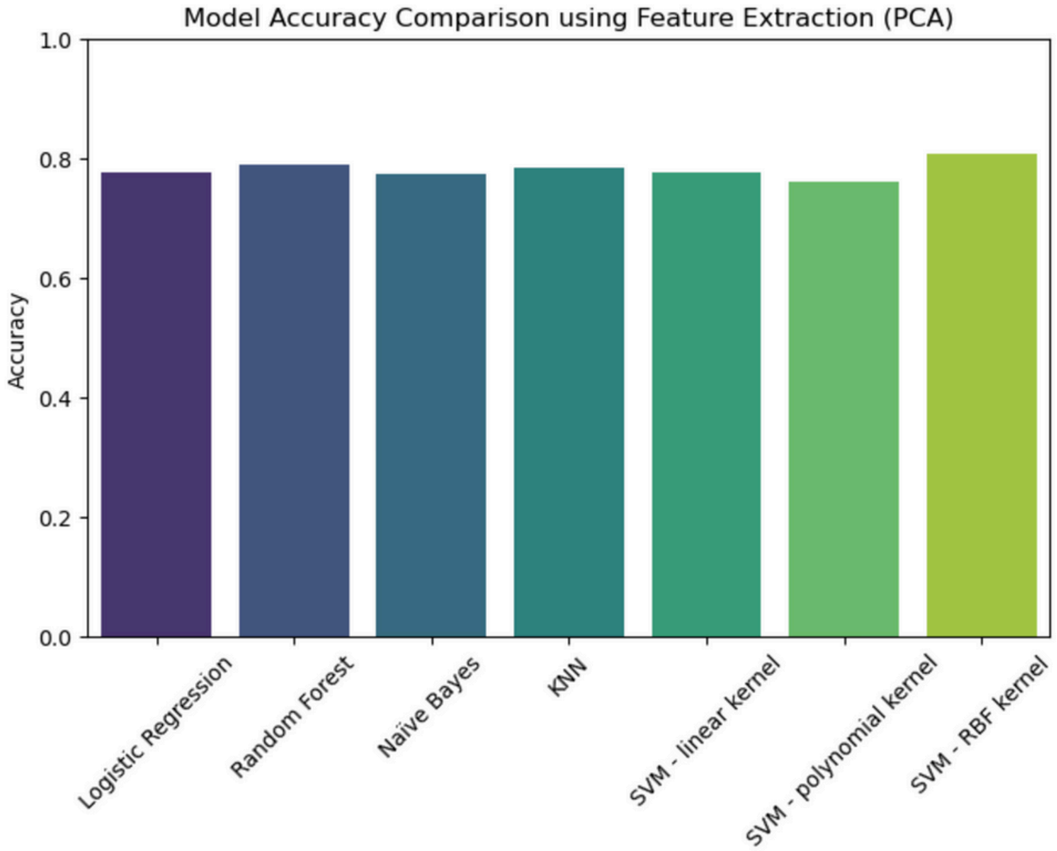
Accuracy Scores of Feature Extraction (PCA)

	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.7985	0.800221	0.7985	0.798463
Random Forest	0.8100	0.810165	0.8100	0.810030
Naïve Bayes	0.7745	0.775279	0.7745	0.774532
KNN	0.7740	0.774171	0.7740	0.774036
SVM - linear kernel	0.8020	0.803178	0.8020	0.801542
SVM - polynomial kernel	0.7700	0.781734	0.7700	0.768453
SVM - RBF kernel	0.7825	0.784649	0.7825	0.782412

Visualization of Accuracy Scores

```
# Extract accuracy scores for visualization
model_names = list(results_sampled.keys())
accuracies = [results_sampled[model]["Accuracy"] for model in model_names]

# Plot accuracy comparison
plt.figure(figsize=(8, 5))
sns.barplot(x=model_names, y=accuracies, palette="viridis")
plt.ylim(0, 1)
plt.ylabel("Accuracy")
plt.title("Model Accuracy Comparison using Feature Extraction (PCA)")
plt.xticks(rotation=45, fontsize=10)
plt.show()
```

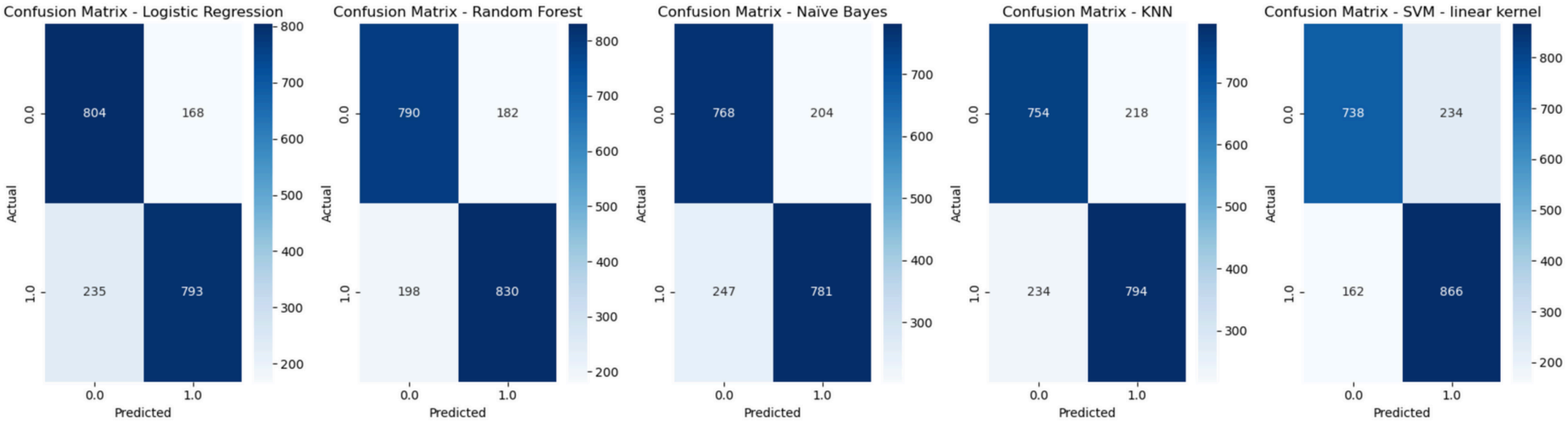




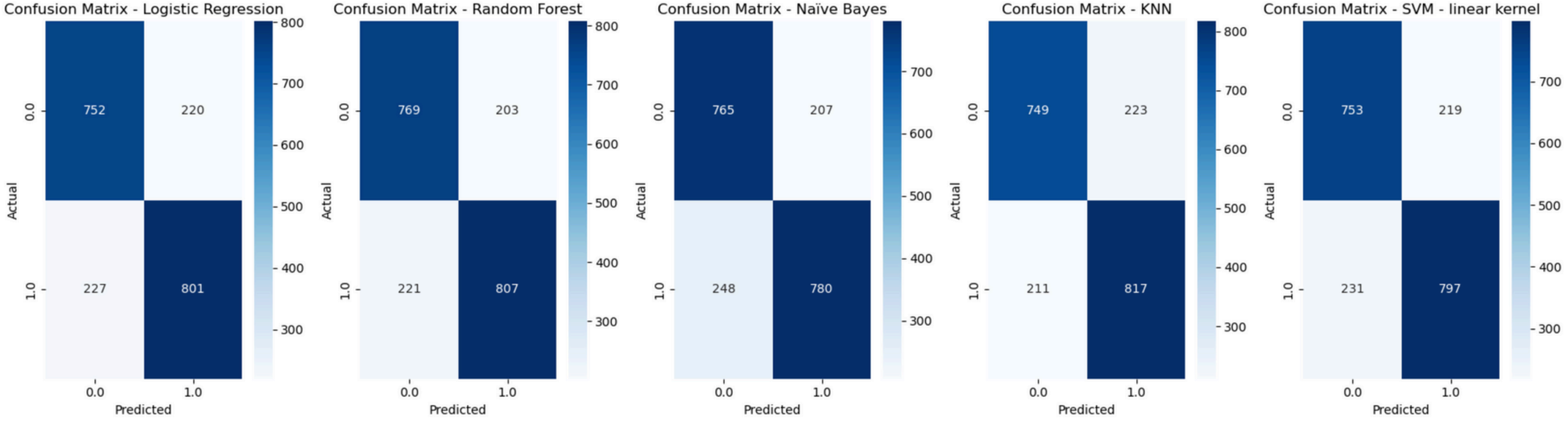
Confusion Matrix

```python
from sklearn.metrics import confusion_matrix

# Selecting the first four models
selected_models12 = list(models12.keys())[:5]

# Creating subplots
fig12, axes12 = plt.subplots(1, 5, figsize=(18, 5))

for i, name12 in enumerate(selected_models12):
    model12 = models12[name12]
    y_pred12 = model12.predict(X_test_sampled12)  # Predictions

    cm12 = confusion_matrix(y_test_sampled12, y_pred12)

    # Heatmap
    sns.heatmap(cm12, annot=True, fmt="d", cmap="Blues", xticklabels=set(y_test_sampled12),
                yticklabels=set(y_test_sampled12), ax=axes12[i])

    axes12[i].set_title(f'Confusion Matrix - {name12}')
    axes12[i].set_xlabel('Predicted')
    axes12[i].set_ylabel('Actual')

# Adjusting layout and showing the plot
plt.tight_layout()
plt.show()
```

Confusion Matrix of Feature Selection (Filter)



Confusion Matrix of Feature Extraction (PCA)



MODEL FITTING WITH ONLY FEATURE SELECTION (FILTER)

We can consider that the Random Forest Model is the best fitted model to our data
with the highest accuracy of 81%,
closely followed by the SVM Model with the Linear Kernel at an accuracy of 80.20 %
and then Logistic Regression at 79.85% accuracy.

MODEL FITTING WITH ONLY FEATURE EXTRACTION (PCA)

We can consider that the SVM Model with the RBF kernel is the best fitted model to our data
with the highest accuracy of 80.75%,
followed by the Random Forest at an accuracy of 78.80 %
and then KNN model at 78.30% accuracy.

The two experiments show that SVM with the RBF kernel performs best as per accuracy for the dataset when only feature extraction through PCA, has been applied for dimensionality reduction, and that the Random Forest classifier performs best in accuracy for the dataset when only feature selection by the Filter method has been applied for dimensionality reduction.

CONCLUSION

- This paper provides a machine learning approach in distinguishing between original and AI-generated images based on texture-based feature extraction and classification techniques.

- Dimension reduction techniques like Feature Selection with 91.03% variance retention, and Principal Component Analysis (PCA) with 95.35% variance retention were effective for the optimization of model performance at lower computational complexity.

- Comparison of various machine learning algorithms—Logistic Regression, Random Forest, Naïve Bayes, KNN and Support Vector Machines (SVM) with various kernels—demonstrated varied efficacy.

- The most promising among them was SVM with an RBF kernel when only PCA was applied, with an accuracy of 80.75%, while the Random Forest model with 81% accuracy was the best model when only the filter method of Feature Selection was applied.

- Logistic Regression, Naïve Bayes, KNN and the SVM models with linear/polynomial kernels reflected mediocre performance.

REFERENCES

[1] Zhong, N., Xu, Y., Li, S., Qian, Z., & Zhang, X. (2023). Patchcraft: Exploring texture patch for efficient ai-generated image detection. arXiv preprint arXiv:2311.12397.

[2] Muthaiah, U., Divya, A., Swarnalaxmi, T. N., & Vidhyasagar, B. S. (2024, December). A Comparative Review of AI-Generated vs Real Images and Classification Techniques. In 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS) (pp. 141-147). IEEE.

[3] Real vs AI-Generated Image Texture Feature Dataset. (2025, February 22). Kaggle.

https://www.kaggle.com/datasets/arkanivasarkar/real-vs-ai-g%20enerated-image-texture-feature-dataset

THANK YOU