Objective:

The objective of this project is to classify bit strings generated by Quantum Random Number Generators (QRNG) and Pseudorandom Number Generators (PRNG) using machine learning techniques. We utilize a stacking ensemble method combining a Convolutional Neural Network (CNN) and a RandomForest model, with Logistic Regression serving as the meta-learner.

Code Documentation:

1. Data Loading:

- **Input File**: The dataset `1000qb_combined_output.txt` contains bit strings and corresponding labels.
- **Bit Strings**: Each row consists of a sequence of binary digits (0s and 1s) representing the generated random numbers.
- **Labels**: Each bit string is labeled as PRNG (0) or QRNG (1).

```
# Load the data

data_filePath = '1000qb_combined_output.txt'

data = pd.read_csv(data_filePath, sep=r'\s+', header=None)


# Display basic data information

print("Data shape:", data.shape)

print(data.head())
```

2. Pre-Processing Techniques:

a. Label Mapping and Cleanup:

- The labels are mapped from their original form to `0` (PRNG) and `1` (QRNG). Any missing labels are filled with the mode.

```
label_mapping = {1: 0, 2: 1}
```

```
labels = labels.map(label_mapping)

labels.fillna(labels.mode()[0], inplace=True)

labels = labels.astype(int)
```

b. Feature Extraction:

- Convert bit strings into arrays of integers (0s and 1s).

```
features_list = []

for bit_string in bit_strings:

    bits = [int(bit) for bit in bit_string]

    features_list.append(bits)

features = np.array(features_list)

print("Features shape after parsing bit strings:", features.shape)
```

c. Data Normalization:

- Standardize the feature values to have a mean of 0 and a standard deviation of 1, which is necessary for effective model training.

```
scaler = StandardScaler()

features = scaler.fit_transform(features)
```

d. Data Splitting:

- The data is shuffled and split into training and testing sets using an 83-17 split. Stratified sampling is used to maintain the distribution of labels across both sets.

python

Copy code

```
X, y = shuffle(features, labels, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.17, stratify=y, random_state=42)
```

---

3. Model Building and Training:

a. CNN Model (Convolutional Neural Network):

- The CNN model consists of two Conv1D layers with ReLU activations and MaxPooling layers. Dropout is applied to reduce overfitting, and a softmax layer is used for classification.

```
def create_cnn_model():

    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)))

    model.add(tf.keras.layers.MaxPooling1D(pool_size=2))

    model.add(tf.keras.layers.Conv1D(filters=128, kernel_size=3, activation='relu'))

    model.add(tf.keras.layers.GlobalMaxPooling1D())

    model.add(tf.keras.layers.Dense(64, activation='relu'))

    model.add(tf.keras.layers.Dropout(0.5))

    model.add(tf.keras.layers.Dense(2, activation='softmax'))


    optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

```
    return model
```

b. Training the CNN:

- The training data is reshaped to fit the CNN input dimensions. The model is trained for 50 epochs with a batch size of 32. The training history (accuracy) is stored for later visualization.

python

Copy code

```
X_train_cnn = X_train.reshape(-1, X_train.shape[1], 1)

X_test_cnn = X_test.reshape(-1, X_test.shape[1], 1)


# Build and Train CNN Model

cnn_model = create_cnn_model()

history = cnn_model.fit(X_train_cnn,
tf.keras.utils.to_categorical(y_train, num_classes=2), epochs=50,
batch_size=32, verbose=1, validation_data=(X_test_cnn,
tf.keras.utils.to_categorical(y_test, num_classes=2)))
```

---

4. Post-Processing Techniques:

a. Stacking Ensemble Model:

- The CNN and RandomForest predictions are stacked to form new features, which are then used to train a Logistic Regression meta-model.

python

Copy code

```
y_train_cnn_pred = np.argmax(cnn_model.predict(X_train_cnn), axis=1)
```

```
y_test_cnn_pred = np.argmax(cnn_model.predict(X_test_cnn), axis=1)


rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train, y_train)


y_train_rf_pred = rf_model.predict(X_train)

y_test_rf_pred = rf_model.predict(X_test)


stacked_train_predictions = np.column_stack((y_train_cnn_pred,
y_train_rf_pred))

stacked_test_predictions = np.column_stack((y_test_cnn_pred,
y_test_rf_pred))


meta_model = LogisticRegression(random_state=42)

meta_model.fit(stacked_train_predictions, y_train)

y_test_meta_pred = meta_model.predict(stacked_test_predictions)
```

b. Evaluation of the Stacked Model:

- The stacked model is evaluated on the test set using accuracy, precision, recall, and F1-score metrics.

python

Copy code

```
accuracy_stacked = accuracy_score(y_test, y_test_meta_pred)

precision_stacked = precision_score(y_test, y_test_meta_pred,
zero_division=0)
```

```
recall_stacked = recall_score(y_test, y_test_meta_pred,
zero_division=0)

f1_stacked = f1_score(y_test, y_test_meta_pred, zero_division=0)


print(f"\nStacking Ensemble Model Test Accuracy:
{accuracy_stacked:.4f}")

print(f"Stacking Ensemble Model Precision: {precision_stacked:.4f}")

print(f"Stacking Ensemble Model Recall: {recall_stacked:.4f}")

print(f"Stacking Ensemble Model F1-Score: {f1_stacked:.4f}")
```

---

5. Visualization of Model Accuracy Over Epochs:

- A histogram is plotted to visualize the CNN model's training and validation accuracy over the epochs.

python

Copy code

```
# Plot CNN training and validation accuracy over epochs

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.title('CNN Accuracy Over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

Summary:

- **Pre-processing techniques**:
  - Label mapping
  - Feature extraction (bit string to integer array)
  - Data normalization
  - Data splitting
- **Post-processing techniques**:
  - Stacking predictions from CNN and RandomForest for meta-learning
  - Evaluation using Logistic Regression as the meta-learner
  - Visualization of model accuracy over epochs

This project demonstrates the combination of CNN and RandomForest models into a stacking ensemble, with Logistic Regression as the meta-model, for improved classification of QRNG and PRNG data. A histogram showing model accuracy over epochs provides further insight into model performance.