Documentation for URL Classification Model using Neural Networks

implementing a classification model to detect different types of URLs (benign, defacement, phishing, malware). The data is processed from a CSV file containing URLs, and a neural network model is used to predict the type of URL based on various extracted features.

Data Preprocessing Steps

1. **Reading the Data:**
   - The data is loaded from the file `"url.csv"`, and its shape and structure are printed.
2. **Feature Engineering:**
   - **URL Length:** A new column is added to represent the length of each URL.
   - **TLD Processing:** The top-level domain (TLD) is extracted using `get_tld()`.
   - **Feature Extraction:**
     - Several features are extracted from the URL such as the number of certain special characters (`@`, `?`, `-`, etc.), whether the URL is using HTTPS, the number of digits and letters, whether the URL uses shortening services, and the number of directories in the path.
   - **Abnormal URL:** A feature is added to flag URLs with abnormal hostname patterns.
   - **Suspicious Words:** A feature is added to detect suspicious words in the URL related to phishing or scam attempts (e.g., “login”, “paypal”).
   - **Hostname Length:** A feature is created for the length of the hostname.
3. **Target Column Transformation:**
   - The target column `type` is mapped to numeric categories for model training (`benign: 0, defacement: 1, phishing: 2, malware: 3`).

Data Split and Preparation

- **Feature Columns (`X`)**: All columns excluding the original URL, type, and domain columns are used as features.
- **Target Column (`y`)**: The target column is the transformed `Category` column.
- **Train-Test Split:**
  - The dataset is split into a training set (80%) and a test set (20%) using `train_test_split()`.
- **PyTorch Tensors:**

- The data is converted into PyTorch tensors for model training. Features are converted to `float32`, while labels are converted to `long`.

Neural Network Model

An advanced feedforward neural network (with 4 layers) is used to classify URLs into one of four categories (benign, defacement, phishing, malware). The model includes:

- **Batch Normalization**: Applied to each hidden layer to stabilize training.
- **Dropout**: Regularization technique used to prevent overfitting (with a rate of 40%).
- **Activation Function**: ReLU (Rectified Linear Unit) is used for the hidden layers.
- **Output Layer**: The final output layer uses the `CrossEntropyLoss` function for multi-class classification.

Model Training

- The model is trained for 25 epochs with an early stopping condition after 10 epochs without improvement in validation loss.
- **Optimizer:** Adam optimizer is used with a learning rate of 0.001 and weight decay for regularization.
- **Learning Rate Scheduler:** A scheduler is applied to adjust the learning rate after every 30 epochs.

Model Evaluation

After training, the model is evaluated on the test set using the following metrics:

- **Accuracy**
- **Precision**
- **Recall**
- **F1 Score**

These metrics are printed for the test set, and a confusion matrix is generated to visualize the classification performance.

Visualization

- **Distributions:** The distribution of features such as URL length and the count of suspicious characters are visualized using seaborn.
- **Confusion Matrix:** A heatmap of the confusion matrix is plotted to assess the model's performance on the test set.

Code Walkthrough

Data Preprocessing

```
python
Copy code
df['url_len'] = [len(url) for url in df.url]  # Add URL length feature
df['domain'] = df['url'].apply(lambda i: process_tld(i))  # Extract TLD (Top-
Level Domain)
df['https'] = df['url'].apply(lambda i: httpSecure(i))  # Check if URL uses H
TTPS
df['digits'] = df['url'].apply(lambda i: digit_count(i))  # Count digits in t
he URL
df['letters'] = df['url'].apply(lambda i: letter_count(i))  # Count letters i
n the URL
df['Shortening_Service'] = df['url'].apply(lambda x: Shortening_Service(x))
# Check for shortening service
df['count_dir'] = df['url'].apply(lambda i: no_of_dir(i))  # Count directorie
s in the URL path
df['sus_url'] = df['url'].apply(lambda i: suspicious_words(i))  # Check for s
uspicious words
```

Model Architecture

```
python
Copy code
class AdvancedNNModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(AdvancedNNModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 256)
        self.fc4 = nn.Linear(256, output_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)  # Output layer without activation
        return x
```

Training Loop

```
python
Copy code
```

```
for epoch in range(epochs):
    # Train model
    model.train()
    # Validation
    model.eval()
```

Model Evaluation

```
python
Copy code
# Final Test Metrics
accuracy = accuracy_score(y_test_tensor.numpy(), y_pred.numpy())
precision = precision_score(y_test_tensor.numpy(), y_pred.numpy(), average='w
eighted')
recall = recall_score(y_test_tensor.numpy(), y_pred.numpy(), average='weighte
d')
f1 = f1_score(y_test_tensor.numpy(), y_pred.numpy(), average='weighted')
```

Confusion Matrix

```
python
Copy code
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'C
lass 1', 'Class 2', 'Class 3'], yticklabels=['Class 0', 'Class 1', 'Class 2',
'Class 3'])
```

Conclusion

This script demonstrates how to process URL data, extract meaningful features, and apply a neural network model for classification tasks. The resulting model can classify URLs into different categories, which is useful for detecting phishing, malware, and other malicious web activities.