Parker Christenson Assignment 6

Anomaly Detection with a Deep Autoencoder

`Instructions`

1. Download the Credit Card Fraud Detection dataset by completing the steps below, and preprocess it as needed for training the deep autoencoder.
2. Go to https://www.kaggle.com/datasets/mlg-ulb/creditcardfraudLinks to an external site. and try to download the dataset.
    - If you don't have access to the dataset, create an account in Kaggle.
    - Once you are logged in, search for "Credit Card Fraud Detection" in the search bar at the top of the page.
    - On the dataset page, click the "Download" button to download the dataset in CSV format.
3. Split the dataset into training, validation, and testing sets.
4. Design and train a deep autoencoder with multiple layers to encode the input data and decode it back to its original form.
5. Use the trained autoencoder to detect anomalies in the test set by comparing the input and output data and calculating reconstruction error.
6. Evaluate the performance of the autoencoder by measuring the accuracy, precision, recall, and F1 score of the anomaly detection.
7. Discuss the limitations and potential applications of deep autoencoders for anomaly detection in credit card transactions and other domains.

In [ ]:

```python
import polars as pl
import numpy as np
import pandas as pd

# autoencoders from pytorch
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset

# sk learns libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# for plotting
import matplotlib.pyplot as plt
import seaborn
```

In [ ]:
```
df = pd.read_csv('creditcard.csv')

df.head()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [ ]:
```
# Separate features and labels
features = df.drop(columns=['Class'])
labels = df['Class']

# Normalize the features
scaler = StandardScaler()
normalized_features = scaler.fit_transform(features)
```

In [ ]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(normalized_features, labels, te
```

In [ ]:
```
# Create a PyTorch Dataset
class FraudDataset(Dataset):
    def __init__(self, data, labels):
        self.data = torch.tensor(data, dtype=torch.float32)
        self.labels = torch.tensor(labels.values, dtype=torch.float32)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]
```

In [ ]:
```
# Loaders and dataset

train_dataset = FraudDataset(X_train, y_train)
test_dataset = FraudDataset(X_test, y_test)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

In [ ]:
```
# model
class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(True),
            nn.Linear(64, 32),
            nn.ReLU(True),
            nn.Linear(32, encoding_dim),
            nn.ReLU(True)
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 32),
            nn.ReLU(True),
            nn.Linear(32, 64),
            nn.ReLU(True),
            nn.Linear(64, input_dim),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

In [ ]:
```
# encoding dim can be adusted to get better results
input_dim = X_train.shape[1]
encoding_dim = 16
model = Autoencoder(input_dim, encoding_dim)
```

In [ ]:
```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [ ]:
```
num_epochs = 50  # Adjust the number of epochs as needed
for epoch in range(num_epochs):
    for data, _ in train_loader:
        output = model(data)
        loss = criterion(output, data)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [1/50], Loss: 0.5072
Epoch [2/50], Loss: 0.5503
Epoch [3/50], Loss: 1.0595
Epoch [4/50], Loss: 0.3465
Epoch [5/50], Loss: 0.6035
Epoch [6/50], Loss: 0.3398
Epoch [7/50], Loss: 0.4453
Epoch [8/50], Loss: 0.2655
Epoch [9/50], Loss: 0.3725
Epoch [10/50], Loss: 0.3139
Epoch [11/50], Loss: 0.2322
Epoch [12/50], Loss: 0.4286
Epoch [13/50], Loss: 0.8521
Epoch [14/50], Loss: 0.2665
Epoch [15/50], Loss: 0.5925
Epoch [16/50], Loss: 0.3181
Epoch [17/50], Loss: 0.5288
Epoch [18/50], Loss: 0.2621
Epoch [19/50], Loss: 0.4673
Epoch [20/50], Loss: 0.2739
Epoch [21/50], Loss: 1.0693
Epoch [22/50], Loss: 0.3406
Epoch [23/50], Loss: 0.3143
Epoch [24/50], Loss: 0.3272
Epoch [25/50], Loss: 0.2748
Epoch [26/50], Loss: 0.9863
Epoch [27/50], Loss: 0.3094
Epoch [28/50], Loss: 0.6825
Epoch [29/50], Loss: 0.3556
Epoch [30/50], Loss: 0.3520
Epoch [31/50], Loss: 0.2767
Epoch [32/50], Loss: 0.5378
Epoch [33/50], Loss: 0.3191
Epoch [34/50], Loss: 1.4361
Epoch [35/50], Loss: 0.2752
Epoch [36/50], Loss: 0.3678
Epoch [37/50], Loss: 0.2717
Epoch [38/50], Loss: 0.4093
Epoch [39/50], Loss: 1.9841
Epoch [40/50], Loss: 0.3230
Epoch [41/50], Loss: 0.6607
Epoch [42/50], Loss: 1.0113
Epoch [43/50], Loss: 0.4028
Epoch [44/50], Loss: 0.5676
Epoch [45/50], Loss: 0.4551
Epoch [46/50], Loss: 0.4222
Epoch [47/50], Loss: 0.4757
Epoch [48/50], Loss: 2.8322
Epoch [49/50], Loss: 0.5660
Epoch [50/50], Loss: 0.5368
```

In [ ]:

```
def get_reconstruction_errors(model, loader):
    model.eval()
    errors = []
    with torch.no_grad():
        for data, _ in loader:
```

```python
            output = model(data)
            loss = nn.MSELoss(reduction='none')(output, data).mean(dim=1)
            errors.append(loss)
    return torch.cat(errors).numpy()
```

In [ ]:
```python
train_errors = get_reconstruction_errors(model, train_loader)
test_errors = get_reconstruction_errors(model, test_loader)
```

In [ ]:
```python
# Determine a threshold for anomalies
threshold = np.percentile(train_errors, 95)
```

In [ ]:
```python
test_anomalies = test_errors > threshold
y_test_numpy = y_test.to_numpy()
accuracy = (test_anomalies == y_test_numpy).mean()

print(f'Anomaly detection accuracy: {accuracy:.4f}')
```

```
Anomaly detection accuracy: 0.9516
```

In [ ]:
```python
# Calculate the reconstruction errors for the training and test sets
train_errors = get_reconstruction_errors(model, train_loader)
test_errors = get_reconstruction_errors(model, test_loader)

# Determine a threshold for anomalies (95th percentile of training errors)
threshold = np.percentile(train_errors, 95)

# Identify anomalies in the test set
test_anomalies = test_errors > threshold

# Convert boolean test anomalies to integers (1 for anomaly, 0 for normal)
test_anomalies = test_anomalies.astype(int)

# Ground truth labels for the test set
y_test_numpy = y_test.to_numpy()

# Calculate the metrics
accuracy = accuracy_score(y_test_numpy, test_anomalies)
precision = precision_score(y_test_numpy, test_anomalies)
recall = recall_score(y_test_numpy, test_anomalies)
f1 = f1_score(y_test_numpy, test_anomalies)

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
```

```
Accuracy: 0.9516
Precision: 0.0311
Recall: 0.8980
F1 Score: 0.0600
```

In [ ]: