SSB GNN Modeli - 1

May 10, 2025

1. VERİYİ GRAF YAPISINA DÖNÜŞTÜRME:

- IoT verisinde her satır bir ölçüm (örnek) olduğuna göre biz her satırı bir düğüm (node) olarak alabiliriz.
- Benzer özelliklere sahip örnekler arasında kenarlar (edge) oluşturarak graf yapısını kurabiliriz.

[14]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Veriyi oku
df = pd.read_csv("C:\\Users\\Lenovo\\Desktop\\OneDrive_2025-05-07\\TON_IoT␣
↪datasets\\Processed_datasets\\Processed_IoT_dataset\\IoT_Fridge.csv")

# Sayısal olmayan sütunları çıkar
features = df.drop(['date', 'time', 'type', 'label', 'temp_condition'], axis=1)

# Etiketleri ayır
labels = df['label']

# Sayısal verileri normalize et
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)
```

[15]:
```
#KNN ile kenar oluşturduğumuz kısım
from sklearn.neighbors import kneighbors_graph
import torch
from torch_geometric.data import Data
from torch_geometric.utils import from_scipy_sparse_matrix

# KNN tabanlı adjacency matrisi oluştur
knn_graph = kneighbors_graph(X_scaled, n_neighbors=10, mode='connectivity',␣
↪include_self=False)

# PyG formatına çevir
edge_index, _ = from_scipy_sparse_matrix(knn_graph)

# Tensorlara çevir
x = torch.tensor(X_scaled, dtype=torch.float)
y = torch.tensor(labels.values, dtype=torch.long)
```

```
# Graph verisi hazır
data = Data(x=x, edge_index=edge_index, y=y)
```

[16]:
```
import torch
from torch_geometric.data import Data
import numpy as np
from sklearn.preprocessing import LabelEncoder
from collections import Counter

# X_scaled verisini tensor yap
x = torch.tensor(X_scaled, dtype=torch.float)

# Label'ları sayıya çevir (0-normal, 1-anomali gibi)
label_encoder = LabelEncoder()
y = torch.tensor(label_encoder.fit_transform(labels), dtype=torch.long)

# Sınıf dağılımını hesapla
class_counts = Counter(data.y.tolist())
print("Sınıf dağılımı:", class_counts)

# Toplam örnek sayısı
total = sum(class_counts.values())

# Her sınıf için ağırlık: toplam / sınıf sayısı
class_weights = [total / class_counts[i] for i in range(len(class_counts))]
print("Sınıf ağırlıkları:", class_weights)

# Edge_index (basitçe her node'u bir sonrakine bağlıyoruz - örnek graf yapısı)
edge_index = []
for i in range(len(x) - 1):
    edge_index.append([i, i + 1])  # Node i --> i+1

edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()

# GNN için Data objesi oluştur
data = Data(x=x, edge_index=edge_index, y=y)

# Kontrol
print(data)
```

```
Sınıf dağılımı: Counter({0: 500827, 1: 86249})
Sınıf ağırlıkları: [1.1722131594342957, 6.806757179793389]
Data(x=[587076, 1], edge_index=[2, 587075], y=[587076])
```

[17]:
```
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
from torch_geometric.loader import DataLoader
```

```
# GCN modeli
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

[8]:
```
# Modeli, optimizer'ı ve loss fonksiyonunu tanımla
model = GCN(in_channels=data.num_node_features, hidden_channels=64,␣
 ↪out_channels=2)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

# Tensor formatına çevir
weights_tensor = torch.tensor(class_weights, dtype=torch.float)

# Ağırlıklı loss fonksiyonu
criterion = torch.nn.CrossEntropyLoss(weight=weights_tensor)

# Train loop
model.train()
for epoch in range(1, 201):
    optimizer.zero_grad()
    out = model(data)
    loss = criterion(out, data.y)
    loss.backward()
    optimizer.step()
    if epoch % 20 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")
```

```
Epoch 20, Loss: 0.4315
Epoch 40, Loss: 0.4191
Epoch 60, Loss: 0.4181
Epoch 80, Loss: 0.4175
Epoch 100, Loss: 0.4174
Epoch 120, Loss: 0.4173
Epoch 140, Loss: 0.4173
Epoch 160, Loss: 0.4173
Epoch 180, Loss: 0.4173
Epoch 200, Loss: 0.4173
```

[18]:
```python
# Tahmin yap ve başarıyı ölç
from sklearn.metrics import classification_report, confusion_matrix

model.eval()
pred = model(data).argmax(dim=1)
correct = (pred == data.y).sum()
acc = int(correct) / len(data.y)
print(f"Accuracy: {acc:.4f}")

# Gerçek etiketler
true_labels = data.y.cpu().numpy()

# Model tahminleri
predictions = model(data).argmax(dim=1).cpu().numpy()

# Raporla
print(confusion_matrix(true_labels, predictions))
print(classification_report(true_labels, predictions, target_names=["Normal",␣
 ↪"Anomali"]))
```

```
Accuracy: 0.8531
[[500827      0]
 [ 86249      0]]

C:\Users\Lenovo\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

      Normal       0.85      1.00      0.92    500827
     Anomali       0.00      0.00      0.00     86249

    accuracy                           0.85    587076
   macro avg       0.43      0.50      0.46    587076
weighted avg       0.73      0.85      0.79    587076


C:\Users\Lenovo\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Lenovo\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
```

TP ve FP 0 veriyor bu da eğitimin yeterli olmadığını modelin yakalayamadığını gösterir.

```
[ ]:
```