SSB Autoencoder Modeli - 1

May 9, 2025

Autoencoder Nedir? - Bir veriyi sıkıştırarak yeniden oluşturmaya çalışan sinir ağıdır. - Anomaliler, bu yeniden oluşturma sırasında yüksek hata üretir biz de bu kısımdan yakalayacağız.

[4]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪auc

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import regularizers
```

[5]:
```
# Veri oku
df = pd.read_csv("C:\\Users\\Lenovo\\Desktop\\OneDrive_2025-05-07\\TON_IoT␣
 ↪datasets\\Processed_datasets\\Processed_IoT_dataset\\IoT_Fridge.csv")
df.drop(['date', 'time', 'type'], axis=1, inplace=True)
df = pd.get_dummies(df, columns=['temp_condition'], drop_first=True)

# Giriş ve hedef
X = df.drop('label', axis=1)
y = df['label']

# Sadece "normal" veriyi eğitim için kullan (denetimsiz yaklaşım!)
X_normal = X[y == 0]

# Ölçekle
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_normal_scaled = scaler.transform(X_normal)

# Train/test ayır
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
 ↪random_state=42)
```

[6]:
```
input_dim = X_train.shape[1]

input_layer = Input(shape=(input_dim,))
encoded = Dense(32, activation="relu")(input_layer)
encoded = Dense(16, activation="relu")(encoded)
decoded = Dense(32, activation="relu")(encoded)
output_layer = Dense(input_dim, activation="linear")(decoded)

autoencoder = Model(inputs=input_layer, outputs=output_layer)
autoencoder.compile(optimizer='adam', loss='mse')
```

[7]:
```
autoencoder.fit(X_normal_scaled, X_normal_scaled,
                epochs=30,
                batch_size=64,
                shuffle=True,
                validation_split=0.2)
```

```
Epoch 1/30
6261/6261 [==============================] - 10s 1ms/step - loss: 0.0086 -
val_loss: 1.4791e-05
Epoch 2/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.7681e-05 -
val_loss: 2.9439e-05
Epoch 3/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.5249e-05 -
val_loss: 1.5363e-05
Epoch 4/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.3856e-05 -
val_loss: 4.7636e-06
Epoch 5/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.3291e-05 -
val_loss: 2.5920e-06
Epoch 6/30
6261/6261 [==============================] - 9s 1ms/step - loss: 9.9119e-06 -
val_loss: 6.8595e-05
Epoch 7/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.1798e-05 -
val_loss: 2.3332e-06
Epoch 8/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.2110e-05 -
val_loss: 2.5250e-05
Epoch 9/30
6261/6261 [==============================] - 9s 1ms/step - loss: 9.5818e-06 -
val_loss: 2.7504e-06
Epoch 10/30
6261/6261 [==============================] - 9s 1ms/step - loss: 1.2966e-05 -
val_loss: 4.7435e-07
Epoch 11/30
```

```
6261/6261 [==============================] - 9s 1ms/step - loss: 7.5158e-06 -
val_loss: 1.2536e-05
Epoch 12/30
6261/6261 [==============================] - 10s 2ms/step - loss: 1.0583e-05 -
val_loss: 2.5636e-07
Epoch 13/30
6261/6261 [==============================] - 9s 1ms/step - loss: 9.8309e-06 -
val_loss: 3.9777e-06
Epoch 14/30
6261/6261 [==============================] - 9s 1ms/step - loss: 8.3223e-06 -
val_loss: 4.5081e-06
Epoch 15/30
6261/6261 [==============================] - 9s 1ms/step - loss: 7.4923e-06 -
val_loss: 1.3557e-05
Epoch 16/30
6261/6261 [==============================] - 9s 1ms/step - loss: 8.2578e-06 -
val_loss: 7.6772e-06
Epoch 17/30
6261/6261 [==============================] - 8s 1ms/step - loss: 8.0121e-06 -
val_loss: 1.1559e-06
Epoch 18/30
6261/6261 [==============================] - 9s 1ms/step - loss: 7.2534e-06 -
val_loss: 1.3357e-06
Epoch 19/30
6261/6261 [==============================] - 9s 1ms/step - loss: 7.6444e-06 -
val_loss: 4.2104e-05
Epoch 20/30
6261/6261 [==============================] - 9s 1ms/step - loss: 7.9546e-06 -
val_loss: 2.6594e-06
Epoch 21/30
6261/6261 [==============================] - 10s 2ms/step - loss: 7.6030e-06 -
val_loss: 9.2807e-06
Epoch 22/30
6261/6261 [==============================] - 9s 1ms/step - loss: 7.2998e-06 -
val_loss: 1.5548e-06
Epoch 23/30
6261/6261 [==============================] - 9s 1ms/step - loss: 6.6364e-06 -
val_loss: 1.4312e-05
Epoch 24/30
6261/6261 [==============================] - 9s 1ms/step - loss: 6.6256e-06 -
val_loss: 1.4114e-06
Epoch 25/30
6261/6261 [==============================] - 9s 1ms/step - loss: 6.8212e-06 -
val_loss: 3.9248e-05
Epoch 26/30
6261/6261 [==============================] - 9s 2ms/step - loss: 8.5298e-06 -
val_loss: 5.1978e-07
Epoch 27/30
```

```
6261/6261 [==============================] - 9s 1ms/step - loss: 5.8413e-06 -
val_loss: 8.8913e-06
Epoch 28/30
6261/6261 [==============================] - 10s 2ms/step - loss: 7.0871e-06 -
val_loss: 3.4332e-06
Epoch 29/30
6261/6261 [==============================] - 9s 1ms/step - loss: 5.5999e-06 -
val_loss: 8.5290e-07
Epoch 30/30
6261/6261 [==============================] - 9s 1ms/step - loss: 5.7296e-06 -
val_loss: 1.3359e-07
```

[7]: `<keras.src.callbacks.History at 0x15fbec7a990>`

[8]:
```python
# Yeniden oluştur
X_test_pred = autoencoder.predict(X_test)

# Yeniden yapılandırma hatasını hesapla (Mean Squared Error)
reconstruction_error = np.mean(np.square(X_test - X_test_pred), axis=1)

# Eşik belirle (manuel veya istatistiksel olarak)
threshold = np.percentile(reconstruction_error, 95)  # üst %5 anomali

# Tahmin yap
y_pred = [1 if err > threshold else 0 for err in reconstruction_error]

# Sonuçları yazdır
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
3670/3670 [==============================] - 4s 1ms/step
[[95689  4338]
 [15871  1518]]
              precision    recall  f1-score   support

           0       0.86      0.96      0.90    100027
           1       0.26      0.09      0.13     17389

    accuracy                           0.83    117416
   macro avg       0.56      0.52      0.52    117416
weighted avg       0.77      0.83      0.79    117416
```

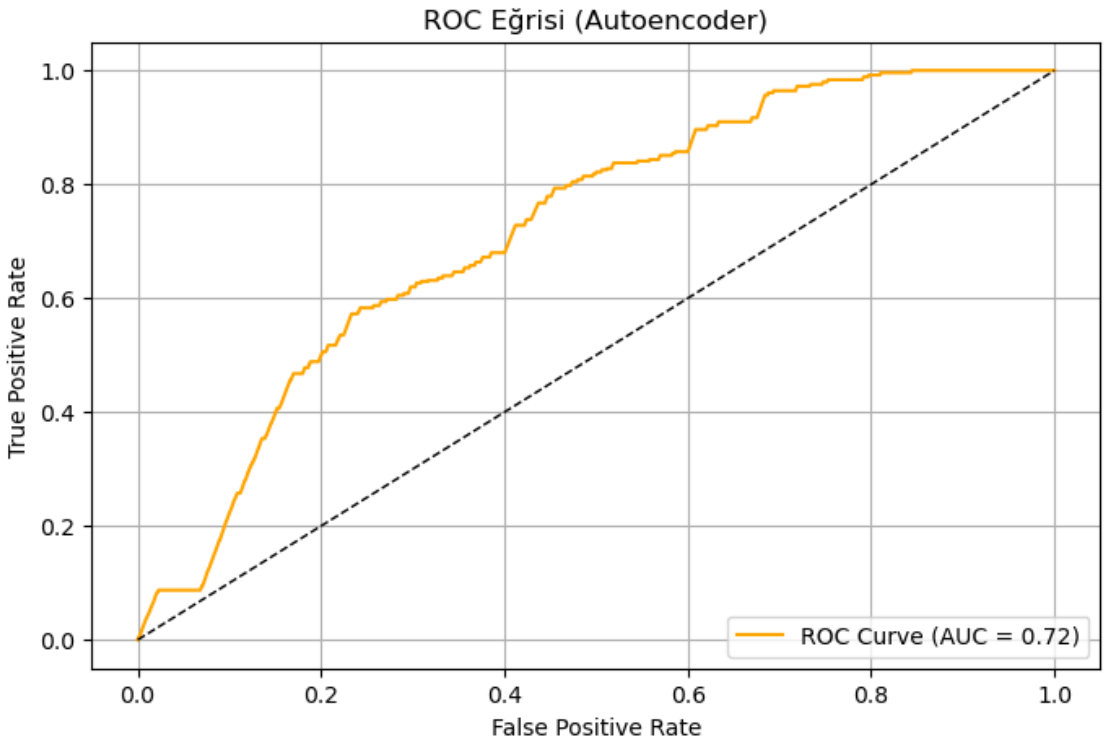
Accuracy %83 - Genel doğruluk yüksek ama bu veri dengesiz olduğu için yanıltıcı olabilir! - Zaten normal veri çok fazla, sadece “normal” deseydi bile accuracy yüksek çıkardı.

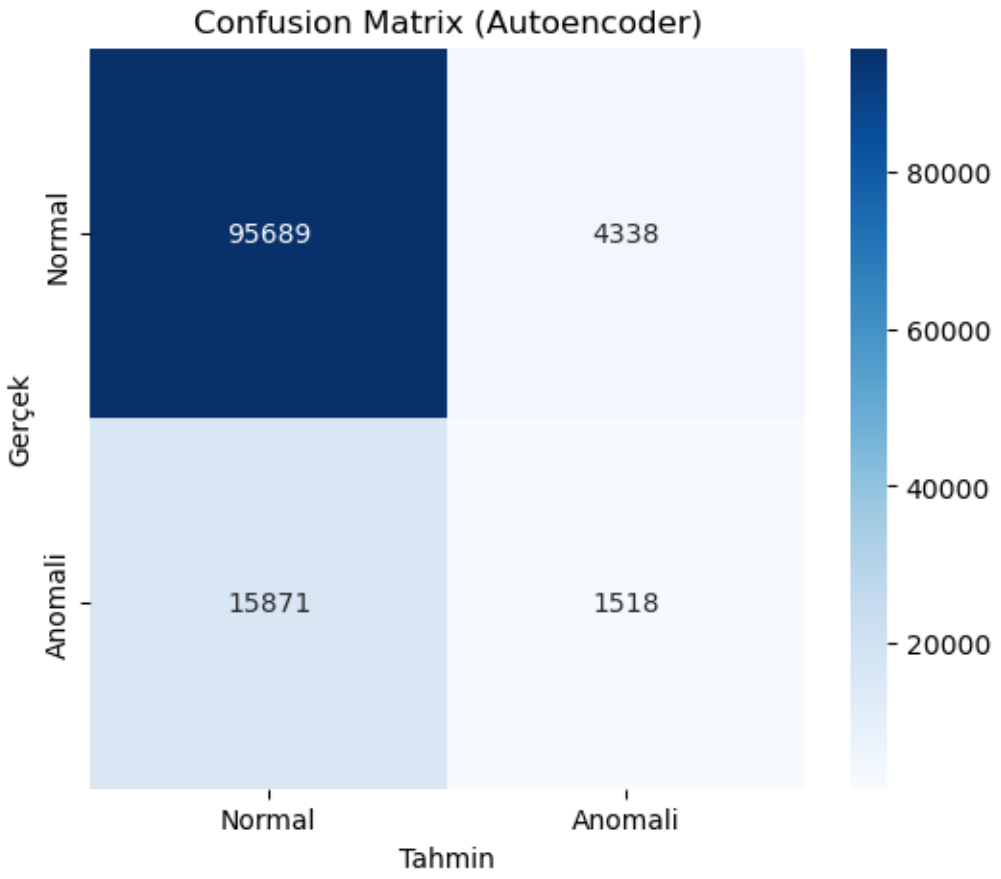
Ne Anladık? - Autoencoder modeli, normal veriyi gayet iyi öğrenmiş. - Ama anomalileri ayırt etme konusunda zayıf kaldı (low recall). - Bu, genelde threshold çok düşük seçilirse olur ya da model daha derin/kompleks olmalıydı.

Bir Sonraki Denemeler için Öneriler: - Threshold'u düşürüp yeniden deneyebilirsin (mesela %90 değil, %85 percentile gibi). - Daha derin bir autoencoder kurabilirsin (daha fazla layer).

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, confusion_matrix

# Autoencoder çıktısını al
X_test_pred = autoencoder.predict(X_test)

# Rekonstrüksiyon hatasını (anomalilik skoru) hesapla
reconstruction_error = np.mean(np.square(X_test - X_test_pred), axis=1)

# ROC eğrisi için
fpr, tpr, thresholds = roc_curve(y_test, reconstruction_error)
roc_auc = auc(fpr, tpr)

# ROC eğrisi çizimi
plt.figure(figsize=(8, 5))
plt.plot(fpr, tpr, color='orange', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Eğrisi (Autoencoder)')
plt.legend(loc='lower right')
plt.grid()
plt.show()

# Confusion matrix için threshold belirle (örnek olarak %95'lik bir eşik)
threshold = np.percentile(reconstruction_error, 95)  # ya da ROC'dan çıkan en␣
 ↪uygun threshold'u seçebilirsin

# Tahminleri binary hale getir
y_pred_class = [1 if err > threshold else 0 for err in reconstruction_error]

# Confusion matrix çizimi
cm = confusion_matrix(y_test, y_pred_class)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (Autoencoder)')
plt.xlabel('Tahmin')
plt.ylabel('Gerçek')
plt.xticks(ticks=[0.5, 1.5], labels=["Normal", "Anomali"])
plt.yticks(ticks=[0.5, 1.5], labels=["Normal", "Anomali"])
plt.show()
```

```
3670/3670 [==============================] - 4s 1ms/step
```


ROC Eğrisi (Autoencoder)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve (AUC = 0.72)



- AUC = 0.72, modelin “anomali” ile “normal” verileri birbirinden ayırt etmede orta seviyede başarılı olduğunu gösteriyor.
- Değer 0.5’ten yüksek olduğu için model rastgele sınıflandırmadan daha iyidir.

- True Positive (TP): 1.518 — Anomalileri doğru tahmin ettiği örnekler.
- False Negative (FN): 15.871 — Anomalileri normal zannetti (tehlikeli!).
- False Positive (FP): 4.338 — Normal verileri yanlışlıkla anomali zannetti.
- True Negative (TN): 95.689 — Normal verileri doğru bildi.

Önemli Nokta: - Recall (1. sınıf için) çok düşük: Anomalileri yakalama oranı %9. - Precision (1. sınıf için) da düşük: Yakalanan anomalilerin doğruluk oranı %26. - Autoencoder modeli bu veri setinde “anomali” sınıfında yüksek hata yapma eğiliminde.

Autoencoder, verideki normal davranışları öğrenmeye çalışır. Bu nedenle anomali sınıfında yeterince temsil olmayan örneklerde düşük başarı gösterebilir. Model daha çok rekonstrüksiyon hatasına dayandığı için doğrudan sınıflandırma problemlerinde (özellikle bu kadar dengesiz sınıflarda) sınırlı performans gösterebilir.

[ ]: