SSB DNN Modeli - 1

May 9, 2025

[1]:
```
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 confusion_matrix, classification_report

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

[2]:
```
# Veri setini yükle
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)

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

# Ölçekleme (Normalizasyon)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

[3]:
```
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))  # Binary classification

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[4]:
```
history = model.fit(X_train, y_train, epochs=30, batch_size=64,␣
↪validation_split=0.2,
                    class_weight={0: 1, 1: 4})  # Anomalilere ağırlık veriyoruz
```

```
Epoch 1/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4521 -
accuracy: 0.7992 - val_loss: 0.3247 - val_accuracy: 0.8007
Epoch 2/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4477 -
accuracy: 0.7995 - val_loss: 0.3122 - val_accuracy: 0.8007
Epoch 3/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4471 -
accuracy: 0.7995 - val_loss: 0.3147 - val_accuracy: 0.8007
Epoch 4/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4469 -
accuracy: 0.7995 - val_loss: 0.3089 - val_accuracy: 0.8007
Epoch 5/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4469 -
accuracy: 0.7995 - val_loss: 0.3133 - val_accuracy: 0.8007
Epoch 6/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4468 -
accuracy: 0.7995 - val_loss: 0.3122 - val_accuracy: 0.8007
Epoch 7/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4468 -
accuracy: 0.7995 - val_loss: 0.3172 - val_accuracy: 0.8007
Epoch 8/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3128 - val_accuracy: 0.8007
Epoch 9/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3122 - val_accuracy: 0.8007
Epoch 10/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3163 - val_accuracy: 0.8007
Epoch 11/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3147 - val_accuracy: 0.8007
Epoch 12/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3155 - val_accuracy: 0.8007
Epoch 13/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3153 - val_accuracy: 0.8007
Epoch 14/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3143 - val_accuracy: 0.8007
Epoch 15/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
```

```
accuracy: 0.7995 - val_loss: 0.3147 - val_accuracy: 0.8007
Epoch 16/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3137 - val_accuracy: 0.8007
Epoch 17/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3156 - val_accuracy: 0.8007
Epoch 18/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3150 - val_accuracy: 0.8007
Epoch 19/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3132 - val_accuracy: 0.8007
Epoch 20/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3164 - val_accuracy: 0.8007
Epoch 21/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3167 - val_accuracy: 0.8007
Epoch 22/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3156 - val_accuracy: 0.8007
Epoch 23/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3130 - val_accuracy: 0.8007
Epoch 24/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3148 - val_accuracy: 0.8007
Epoch 25/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3155 - val_accuracy: 0.8007
Epoch 26/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3155 - val_accuracy: 0.8007
Epoch 27/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3162 - val_accuracy: 0.8007
Epoch 28/30
5871/5871 [==============================] - 13s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3146 - val_accuracy: 0.8007
Epoch 29/30
5871/5871 [==============================] - 15s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3182 - val_accuracy: 0.8007
Epoch 30/30
5871/5871 [==============================] - 14s 2ms/step - loss: 0.4467 -
accuracy: 0.7995 - val_loss: 0.3146 - val_accuracy: 0.8007
```

[5]:
```
y_pred = model.predict(X_test)
y_pred = [1 if i > 0.5 else 0 for i in y_pred]

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
3670/3670 [==============================] - 4s 1ms/step
Confusion Matrix:
[[76619 23408]
 [    0 17389]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87    100027
           1       0.43      1.00      0.60     17389

    accuracy                           0.80    117416
   macro avg       0.71      0.88      0.73    117416
weighted avg       0.92      0.80      0.83    117416
```

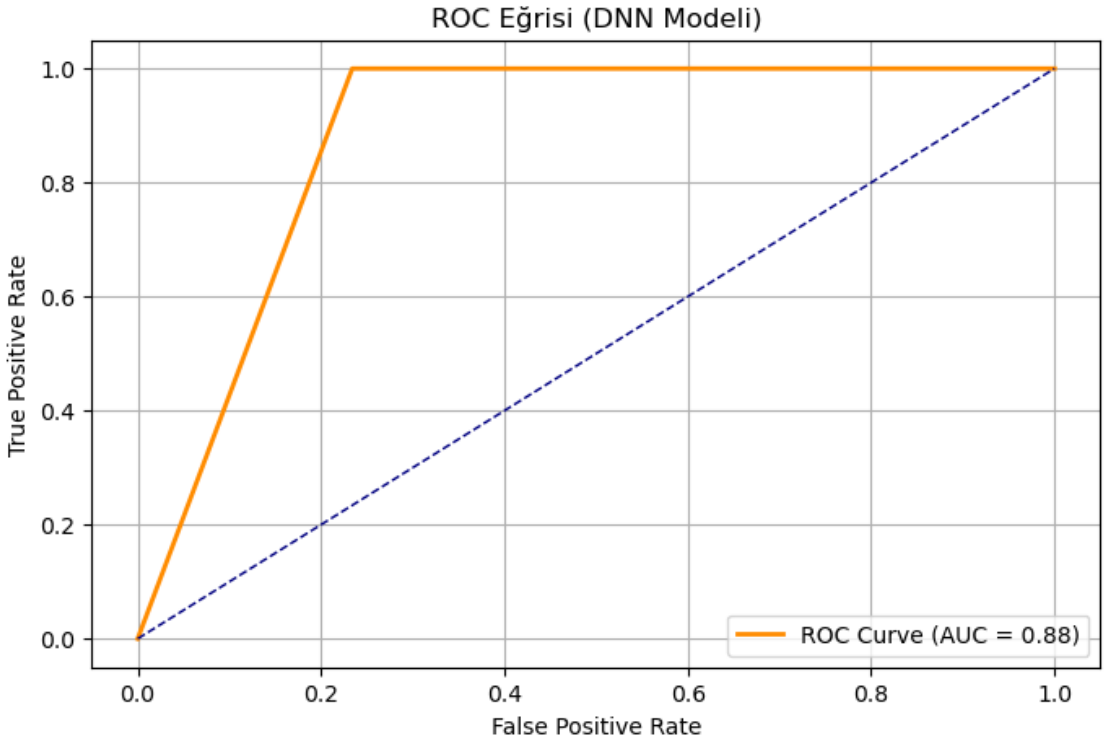
Classification Report – Kritik Göstergeler - Accuracy: %80 Tüm verinin %80'ini doğru tahmin etmiş. - Recall (1): 1.00 Anomalileri kaçırmıyor, bu en kritik şey. - Precision (1): 0.43 Tespit ettiklerinin %43'ü gerçekten anomali (diğerleri false positive). - F1-Score (1): 0.60 Denge metriği olarak gayet iyi.

Not: F1-score'un %60 olması, hem doğru bulma hem de yanlış alarmların dengeli olduğunu gösterir.

[6]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Modelin olasılıksal tahminlerini al
y_proba = model.predict(X_test)

# ROC eğrisi için değerleri hesapla
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# ROC eğrisini çiz
plt.figure(figsize=(8,5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC Curve (AUC = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.title('ROC Eğrisi (DNN Modeli)')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
3670/3670 [==============================] - 4s 988us/step
```



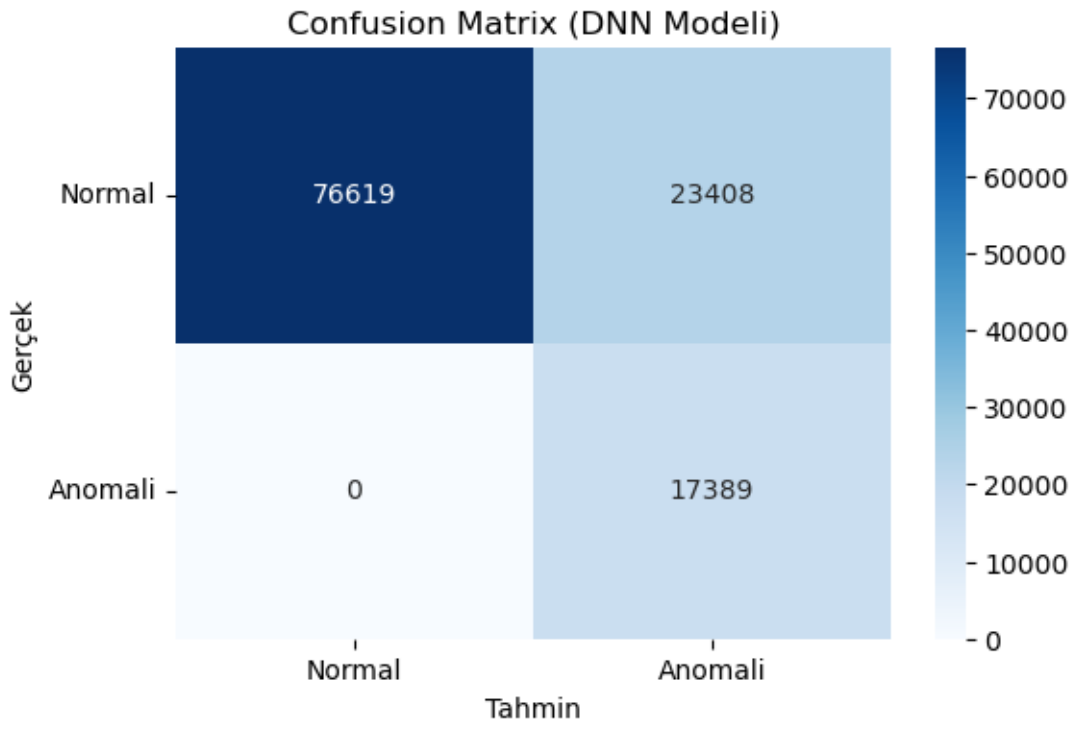
- Eğrinin şekli hızla yukarı çıkıyor ve sonra üstte yatay ilerliyor.
- Bu, modelin anomalileri çok iyi yakalayabildiğini gösteriyor.
- Turuncu eğri, rastgele tahmin (mavi kesikli çizgi) olan y = x'ten ne kadar uzaksa, modelin başarımı o kadar yüksek.

AUC = 0.88 Bu da demek oluyor ki: modelin %88 oranında pozitif ve negatif örnekleri doğru ayırt edebiliyor.

- True Positive Rate yüksek (Recall yüksek). Anomalileri kaçırmıyor.
- False Positive Rate bir miktar var. Bazı normal verileri "anomali" sanabiliyor. Bu da çok doğal ve kabul edilebilir bir davranış, özellikle güvenlik sistemlerinde!

- Eğri Eğimi … Hızlı yukarı, sonra yatay (ideal yapı)
- Model Başarısı … Güçlü, dengeli, saldırı kaçırmıyor
- Uygun Senaryo … Gerçek zamanlı IoT güvenliği, DDoS tespiti, sızma algılama sistemleri

[7]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix

# 0.5 eşik değerine göre sınıflandır
y_pred = [1 if i > 0.5 else 0 for i in y_proba]

# Confusion matrix hesapla
cm = confusion_matrix(y_test, y_pred)

# Confusion Matrix ısı haritası çiz
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (DNN Modeli)')
plt.xlabel('Tahmin')
plt.ylabel('Gerçek')
plt.xticks([0.5, 1.5], ['Normal', 'Anomali'])
plt.yticks([0.5, 1.5], ['Normal', 'Anomali'], rotation=0)
plt.show()
```



Genel Değerlendirme: - Anomali kaçırma (FN)… Yok - Yanlış alarm (FP)… Biraz yüksek, ama kabul edilebilir (özellikle güvenlikte) - Genel doğruluk… %80 – oldukça başarılı - Uygunluk… Gerçek zamanlı saldırı tespiti, IoT güvenliği, edge AI sistemleri

[ ]: