

Süper Bilgisayar Loglarından Arıza Tahmini: Isolation Forest ile Gözetimsiz Anomali Tespiti

743 MB'lık ham log dosyasından, hiçbir etiket görmeden sistem arızalarını öngören bir erken uyarı sistemi—ve denetimli derin öğrenme modellerinin beklenmedik yenilgisi.*

— -

Her gün milyonlarca sunucu, konsol loglarında binlerce satır mesaj üretir: donanım uyarıları, dosya sistemi hataları, ağ sorunları... Çoğu zararsızdır. Ama bazıları, saatler sonra gerçekleşecek bir arızanın ilk sinyalidir.

Peki bu sinyalleri **hiçbir arıza etiketi görmeden**, yalnızca “normalin” ne olduğunu öğrenerek yakalayabilir miyiz?

Bu yazıda, Blue Gene/L (BGL) süper bilgisayarının logları üzerinde geliştirdiğim uçtan uca bir anomali tespit projesini anlatacağım: ham log satırlarının yapılandırılmış olaylara dönüştürülmesinden, kayan pencerelerle öznitelik çıkarımına ve Isolation Forest modelinin eğitilmesine kadar. Yol boyunca ilginç bir bulguyla da karşılaşacağız: aynı veri üzerinde eğitilen BiLSTM tabanlı denetimli modeller, gözetimsiz yaklaşımı geçemedi.

— -

Neden BGL Veri Kümesi?

Log tabanlı arıza tahmini araştırmalarının standart kıyaslama veri kümelerinden biri olan **BGL (Blue Gene/L)**, Los Alamos National Laboratory bünyesinde bulunan ve 131.072 işlemciye sahip dünyanın en hızlı süper bilgisayarlarından biri tarafından üretilen konsol loglarından oluşuyor.

Her log satırı; zaman damgası, uyarı/durum kodu ve düz metin mesajı içeriyor. Veri kümesi; donanım arızaları, dosya sistemi hataları, ağ sorunları ve kullanıcı seviyesi hatalar gibi çok çeşitli olay tiplerini barındırıyor.

Projede kullandığım bölmede rakamlar şöyle:

- **Ham dosya:** ~743 MB
- **Toplam log satırı:** 3.017.161
- **Farklı olay tipi:** 5.000
- **Alert oranı:** ~%8,4 (sınıflar arasında belirgin dengesizlik)

`KERN`, `APP`, `RAS` gibi ön eklerle başlayan satırlar “alert”—yani gerçek bir arıza/uyarı durumu—olarak etiketleniyor. Ancak bu etiketleri modeli eğitmek için değil, yalnızca sonucu doğrulamak için kullanacağız.

— -

Adım 1: Ham Logları Anlamlandırmak—Drain Algoritması

Ham log satırları yapısal olmayan düz metinlerdir; doğrudan makine öğrenmesi modeline veremezsiniz. Önce her satırın sabit bir ****mesaj şablonuna**** dönüştürülmesi gerekiyor:

Bu amaçla, çevrimiçi çalışan ve hesaplama verimliliği yüksek olan **Drain** algoritmasını (He vd., 2017) tercih ettim. Drain, log satırlarını uzunluklarına göre gruplandırıyor, ardından token bazlı benzerliğe göre sabit derinlikli bir ağaç yapısı içinde yönlendiriyor. Benzer satır gruplarına benzersiz bir ``event_id`` atanıyor.

Kullandığım temel parametreler:

- ``similarity_threshold = 0.5``
- ``depth = 4``
- ``max_children = 100``
- ``max_log_len = 128``

Ayrıştırma sonunda her satır; zaman damgası, ikili alert etiketi, 1–5000 arası bir ``event_id`` ve şablon metnine indirgenmiş hâliyle yapılandırılmış bir CSV'e dönüşüyor.

— -

Adım 2: Zamandan Öznitelik Çıkarmak—Kayan Pencereler

Log verisi doğası gereği bir zaman serisi. Bu yüzden veriyi sabit uzunlukta zaman pencerelerine böldüm ve her pencereyi tek bir gözlem olarak ele aldım:

- **Pencere genişliği:** 15 dakika
- **Kaydırma adımı:** 15 dakika (örtüşmesiz)
- **Tahmin ufku (horizon):** pencere kapanışını izleyen 15 dakika

Etiketleme mantığı basit: Bir pencerenin **izleyen çeyrek saat içinde** en az bir alert gerçekleştiyse etiket 1, aksi hâlde 0. Model böylece şu soruya yanıt vermeye çalışıyor: ****"Önümüzdeki 15 dakikada bir arıza meydana gelecek mi?"****

Her pencere için çıkardığım öznitelikler:

- **Olay frekans histogramı** (5.002 boyutlu vektör)
- **Alert oranı**
- **Benzersiz olay sayısı**
- **Sekans uzunluğu**
- **En sık 10 olayın yoğunlaşması** (top-10 concentration)
- **Shannon entropisi:** $H = -\sum p \cdot \log_2(p)$
- **Maksimum tek olay frekansı**

Sonuç: Her pencere için **5.008 boyutlu** bir öznitelik vektörü ve toplamda **5.957 pencere**. Bunların yalnızca **%4,48'i (267 pencere)** pozitif—ciddi bir sınıf dengesizliğiyle karşı karşıya olduğumuzu baştan söyleyeyim.

— -

Adım 3: Isolation Forest—Anomalileri İzole Etmek

Geleneksel denetimli modellerin aksine **Isolation Forest** (Liu vd., 2008), arıza etiketlerine ihtiyaç duymaz. Temel fikri çok zarif:

> Anomaliler azınlıktadır ve diğer gözlemlerden uzaktır. Bu yüzden rastgele bölmelerle **daha az adımda izole edilirler.**

Algoritma, rastgele seçilen öznitelikler ve rastgele bölme değerleriyle yüzlerce izolasyon ağacı (iTree) oluşturur. Bir gözlemin anomali skoru, ağaçlar arasındaki ortalama yol uzunluğundan hesaplanır:

$$s(x) = 2^{(-E[h(x)] / c(n))}$$

Skor 1'e yaklaşırsa gözlem anomali, 0,5'in altındaysa normal kabul edilir.

Neden Isolation Forest?

1. **Etiket gerektirmez**—gerçek sistemlerde arıza etiketleri çoğu zaman yoktur; model etiketsiz yeni sistemlere taşınabilir.
2. **Yüksek boyutlu ve seyrek vektörlerle** verimli çalışır.
3. **Doğrusal olmayan ilişkileri** yakalar ve paralel hesaplamaya elverişlidir.

Eğitim öncesi öznitelikleri `StandardScaler` ile standardize ettim (ortalama 0, std 1). Kritik detay: Ölçekleyici **yalnızca eğitim bölümünde** öğrenildi, test bölümüne aynı dönüşüm uygulandı.

Zaman serisi olduğundan eğitim/test ayrımını da **kronolojik** yaptım—shuffle yok. İlk %80 (4.765 pencere) eğitim, son %20 (1.192 pencere) test.

Model hiperparametreleri:

- `n\_estimators = 200`
- `contamination = 0.06` (beklenen anomali oranı)
- `random\_state = 42`, `n\_jobs = -1`

— -

Sonuçlar: Etiketsiz Öğrenme Ne Kadar İşe Yarar?

Test bölümündeki 1.192 pencereden yalnızca 22'si gerçekten pozitif. Isolation Forest'ın sonuçları:

Metrik	Değer
Accuracy	0,9706
Precision (anomali)	0,24
Recall (anomali)	0,27
F1-skoru (anomali)	0,26
ROC-AUC	**0,81**

Bu sayılar ne anlatıyor?

İyi haber: Hiçbir etiket görmeden, ROC-AUC değeri 0,81 ile anomali skorları pozitif pencereleri normal pencerelerden belirgin biçimde ayırt edebildi. Model “istatistiksel olarak sıra dışı” pencereleri gerçekten yakalıyor.

Dürüst olmak gerekirse: Precision 0,24—yani üretilen her ~4 alarmdan 3'ü yanlış. Ama bunu bağlama koymak lazım: Gözetimsiz paradigmada model “arıza gerçekleşecek pencere”yi değil, “normalden sapan pencere”yi arar. Düşük precision, %1,85'lik pozitif oranla birleşince kaçınılmaz bir sonuç.

Eşik oyunu: Karar eşliğini düşürdüğümüzde recall'u %91–100 bandına çıkarabiliyoruz—bedeli alarm sayısının artması. Erken uyarı sistemlerinde genelde yüksek recall daha kritiktir: Bir arızayı kaçırmak, yanlış alarmdan çok daha pahalıdır.

Pencere genişliği ödünleşimi (trade-off)

İlginç bir deney daha: Önceki 30 dakikalık pencere yapılandırmasıyla (ROC-AUC 0,80; recall %57,7) kıyasladığımda, 15 dakikalık pencereler ROC-AUC'yi hafif artırdı (0,81) ama recall'u belirgin düşürdü (%27,3).

Kısa pencereler anomaliyi dar bir ufkua sıkıştırarak skor ayırt ediciliğini artırıyor; ama pozitif örnek sayısı azaldığı için duyarlılık düşüyor. **Pencere genişliği, uygulama hedefine göre yapılması gereken kritik bir tasarım kararı.**

— -

Plot Twist: Denetimli Modeller Neden Kaybetti?

Karşılaştırma amacıyla, literatürün güçlü mimarilerinden **LogRobust** (BiLSTM + Multi-Head Attention, Zhang vd., 2019)'u farklı kayıp fonksiyonlarıyla eğittim. Sonuçlar:

Model	ROC-AUC	Precision	F1
BiLSTM (CrossEntropy)	0,44	0,00	0,00
BiLSTM (BCEWithLogitsLoss)	0,71	0,00	0,00
BiLSTM (Focal Loss)	0,37	0,00	0,00
Isolation Forest	0,81	0,24	0,26

Üç konfigürasyonun tamamında precision ve F1 sıfır: Modeller, karar eşğinde tek bir pozitif örneği bile isabetle seçemedi. Neden?

1. **Aşırı sınıf dengesizliği:** Test bölmesinde pozitif oran %1,85. Derin mimariler sağlam bir karar sınırı öğrenebilmek için görece dengeli ve geniş pozitif örnek kümeleri ister; yoksa çoğunluk sınıfını tahmin etmeyi “öğrenir”.

2. **Veri hacmi ↔ mimari karmaşıklığı oranı:** Multi-head attention içeren bir BiLSTM'in parametre sayısı, 4.765 pencerele eğitim kümesini fazlasıyla aşıyor. Overfitting/underfitting riski yüksek.

3. **Kayıp fonksiyonu kalibrasyonu:** BCE, ciddi dengesizlikte gradyanın çoğunluk sınıfı

tarafından baskılanmasına yol açar. Focal Loss ise gamma ve alpha parametrelerinin özenle ayarlanmasına muhtaç—ROC-AUC'nin 0,37'ye (rastgele tahminin altına!) gerilemesi bundan.

Önemli nüans: Bu başarısızlığın sebebi denetimli yöntemin doğası değil, **veri ve kalibrasyon koşulları**. Uygun örnekleme (SMOTE, ağırlıklı örnekleme), öznitelik temsili ve eşik kalibrasyonu bu modellerin başarımlarını artırılabilir.

— -

Ne Öğrendim?

Bu deneyin bana kazandırdığı üç ana çıkarım:

1. Etiket yoksa paniğe gerek yok. Isolation Forest gibi gözetimsiz yöntemler, sadece etiket maliyetini ortadan kaldırmakla kalmıyor; pozitif örneklerin son derece sınırlı olduğu erken aşama veri kümelerinde de rekabetçi bir temel (baseline) oluşturuyor.

2. ROC-AUC tek başına yetmez, eşik stratejisi operasyonel karardır. Aynı model, eşik tercihinin göre “hassas ama gürültülü” ya da “seçici ama kaçırıcı” bir sisteme dönüşebilir. Erken uyarı senaryolarında bu tercih işletmeye aittir.

3. Basit, iyi tasarlanmış bir pipeline karmaşık bir mimariyi yenebilir.** Drain + kayan pencere + istatistiksel öznitelikler + Isolation Forest zinciri, aynı veride BiLSTM'i açık ara geçti.

— -

Gelecek Adımlar

- `contamination` parametresinin zaman içindeki anomali oranına göre dinamik belirlenmesi
- Meta özniteliklere pencereler arası zamansal trend bilgisi eklenmesi
- SMOTE / ağırlıklı örnekleme ve eşik kalibrasyonu ile denetimli modellerin yeniden değerlendirilmesi
- Gözetimsiz + yeniden kalibre edilmiş denetimli modellerin hibrit (ensemble) birleşimi

— -

Kaynaklar

1. Olinas, R. K., & Barton, T. (2007). *Blue Gene/L System Logs Dataset.* Los Alamos National Laboratory—usenix.org/cfdr-data
2. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). *Isolation Forest.* IEEE ICDM, 413–422.
3. He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). *Drain: An Online Log Parsing Approach with Fixed Depth Tree.* IEEE ICWS, 33–40.
4. Zhang, X., et al. (2019). *Robust Log-Based Anomaly Detection on Unstable Log Data.* ESEC/FSE, 807–817.
5. Lin, T.-Y., et al. (2017). *Focal Loss for Dense Object Detection.* ICCV, 2980–2988.
6. Xu, W., et al. (2009). *Detecting Large-Scale System Problems by Mining Console Logs.* SOSP'09.

