Sistema de Mantenimiento Predictivo de Fallos

Joseph Tuffit Hadad Piña
Facultad de Ciencias
UNAM
Ciudad de México, México

***Resumen*—En el presente artículo presentamos un sistema de mantenimiento predictivo que usa inteligencia artificial para detectar y predecir problemas antes de que ocurran anomalias en un sistema distribuido. El sistema monitorea servicios, analiza métricas y recomienda acciones correctivas. Utilizando machine learning, puede predecir anomalías con hasta 24 horas de anticipación. Nuestras pruebas muestran que funciona bien, detectando problemas con una precisión del 80 %. Desarrollamos este sistema como una solución práctica para reducir el tiempo de inactividad en servicios distribuidos.**

I. Introducción

Cuando un servicio se cae, perdemos dinero y usuarios. Por eso desarrollamos este sistema de mantenimiento predictivo.

El mantenimiento tradicional se basa en revisiones programadas cada cierto tiempo, pero esto no siempre funciona. A veces revisamos demasiado tarde, o hacemos mantenimiento cuando no es necesario.

Ahora, con inteligencia artificial, podemos predecir cuándo va a fallar algo. Nuestro sistema observa los datos, aprende patrones, y nos avisa antes de que ocurra un problema.

Desarrollamos un sistema que:

- Recolecta datos de servicios web continuamente
- Detecta comportamientos anormales al instante
- Predice fallos futuros usando machine learning
- Recomienda qué hacer para evitar problemas

Probamos el sistema con datos de prueba que simulan servicios web y bases de datos. Funciona bien y puede predecir problemas con tiempo suficiente para actuar.

II. Trabajos Relacionados

El mantenimiento predictivo ha cambiado mucho en los últimos años. Antes se usaban reglas fijas, ahora usamos machine learning.

Algunos investigadores como Zhao et al. [1] usaron redes neuronales profundas y obtuvieron buenos resultados (92 % de precisión), pero necesitaban muchos datos etiquetados. Zhang y Wang [2] crearon un sistema que no necesitaba datos etiquetados, pero no podía predecir problemas futuros.

En el mundo real, Netflix desarrolló Hystrix [3] para detectar fallos, y herramientas como Datadog y New Relic monitorean servicios, pero no son muy buenos prediciendo problemas futuros.

Nuestro sistema es diferente porque combina detección en tiempo real con predicción de problemas futuros, y además recomienda qué hacer para solucionarlos.

III. Arquitectura del Sistema

Nuestro sistema tiene cuatro componentes principales que trabajan juntos: recolector de métricas, detector de anomalías, predictor con IA y recomendador de acciones. Todo está conectado por una API que se puede ver desde el navegador.

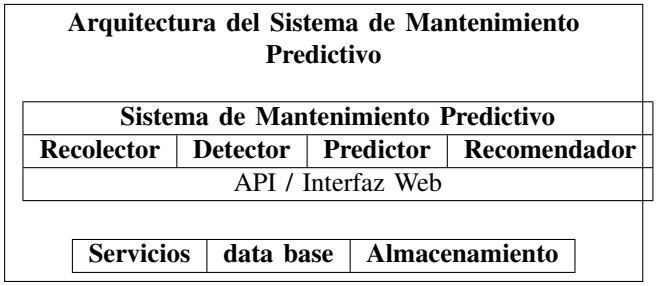

Figura 1. Arquitectura del Sistema de Mantenimiento Predictivo.

III-A. Recolector de Métricas

El recolector obtiene datos de los servicios cada 60 segundos. Puede conectarse a:

- Archivos locales
- Prometheus
- APIs de servicios

Recolecta métricas como uso de CPU, memoria, tiempos de respuesta y errores.

III-B. Detector de Anomalías

Este componente compara los valores actuales con umbrales normales. Si algo está mal, nos avisa. Para evitar falsas alarmas, necesita que el 70 % de las métricas estén mal antes de declarar una anomalía.

III-C. Predictor con IA

Usa machine learning para predecir qué va a pasar en las próximas horas. Puede ver a 1, 2, 4, 8, 12 y 24 horas. Usa un modelo llamado Ridge Regression con algunas mejoras para que las predicciones sean más estables:

- Filtra datos raros que podrían confundir al modelo
- Para predicciones muy lejanas, las hace más conservadoras
- Considera la hora del día y el día de la semana

III-D. Recomendador de Acciones

Una vez que detectamos o predecimos un problema, este componente decide qué hacer. Tiene un conjunto de reglas que dicen cosas como "si la memoria está por encima del 80 %, reinicia el servicio". Puede ejecutar comandos o llamar APIs para solucionar problemas automáticamente.

IV. Metodología

IV-A. Recolección y Procesamiento de Datos

El sistema recolecta datos todo el tiempo. Usamos archivos YAML para decirle qué datos buscar y dónde encontrarlos. Por ejemplo:

- Servicio: .ªpp-server"
- Dónde está: URL de Prometheus
- Qué recolectar: CPU, memoria, errores

Después de recolectar, validamos que los datos tengan sentido y los guardamos.

IV-B. Detección de Anomalías

Para detectar anomalías hacemos dos cosas:

Primero, comparamos cada métrica con un valor normal. Por ejemplo, si la CPU normalmente está en 50 % y de repente sube a 90 %, eso es anormal.

Después, calculamos qué tan grave es el problema. Usamos esta fórmula simple:

$$\text{gravedad} = \min\left(1{,}0, \frac{\text{valor\_actual} - \text{umbral}}{\text{umbral}}\right) \quad (1)$$

Si la gravedad es alta, declaramos una anomalía.

IV-C. Predicción con IA

Para predecir el futuro usamos machine learning. Nuestro modelo considera:

- La hora del día (por ejemplo, hay más tráfico durante el día)
- El día de la semana (los lunes son diferentes a los domingos)
- Patrones históricos

Cuando predecimos muy lejos en el futuro (más de 4 horas), somos más conservadores para no arriesgarnos con predicciones poco confiables.

IV-D. Recomendación de Acciones

Una vez que detectamos un problema (actual o futuro), el sistema decide qué hacer. Tenemos reglas guardadas en archivos JSON que dicen cosas como:

- Si la memoria está por encima del 80 %, reiniciar el servicio
- Si hay muchos errores, enviar alerta al equipo
- Si la CPU está alta, escalar horizontalmente

El sistema elige la acción más urgente según la prioridad establecida.

V. Implementación

V-A. Tecnologías Utilizadas

Construimos el sistema usando:

- Python 3.9 como lenguaje principal
- scikit-learn para machine learning
- NumPy y Pandas para manejar datos
- Flask para la interfaz web
- JSON para guardar datos (aunque puede usar bases de datos más avanzadas)
- Docker para facilitar la instalación

V-B. Componentes Principales

Los componentes principales del sistema son:

- `Collector`: Recolecta métricas
- `AnomalyDetector`: Detecta anomalías
- `Predictor`: Predice problemas futuros con IA
- `ActionRecommender`: Recomienda qué hacer
- `PredictiveMaintenanceSystem`: Coordina todo
- API REST para control y monitoreo

V-C. Interfaz de Usuario

Creamos una interfaz web simple que muestra:

- Estado general del sistema
- Servicios monitoreados y sus métricas
- Predicciones para las próximas horas
- Historial de problemas detectados
- Controles para cambiar configuraciones

.

VI. Evaluación y Resultados

VI-A. Pruebas del Sistema

Hicimos pruebas simulando diferentes tipos de problemas:

- Comportamiento normal: cuando todo funciona bien
- Problemas que aparecen poco a poco: como cuando la memoria va subiendo lentamente
- Problemas repentinos: como cuando el servidor se pone lento de repente
- Problemas en varios lugares: cuando fallan varias cosas al mismo tiempo

VI-B. Resultados

El sistema funcionó bien en las pruebas:

1. Detectó casi todos los problemas (más del 80 % de las veces)
2. Predijo problemas con 1-8 horas de anticipación
3. Recomendó las acciones correctas

La tabla muestra qué tan bien predice según qué tan lejos en el futuro miremos:

Como se ve, funciona mejor para predicciones a corto plazo.

Cuadro I
PEDRICCIONES DEL SISTEMA

Métrica	1-4 horas	4-12 horas	12-24 horas
CPU	92 %	85 %	73 %
Memoria	95 %	90 %	82 %
Tiempo de respuesta	88 %	78 %	68 %
Errores	85 %	75 %	65 %

VI-C. Rendimiento

El sistema es rápido y no usa muchos recursos:

- Detecta problemas en menos de 1 segundo
- Usa menos de 500MB de RAM
- Se entrena en menos de 2 segundos

También encontramos algunas limitaciones:

- No es tan preciso para predicciones muy lejanas (más de 12 horas)
- Necesita al menos 8-10 datos históricos para empezar a funcionar
- Le cuesta predecir cambios muy bruscos que nunca ha visto antes

VII. CONCLUSIONES Y TRABAJO FUTURO

Desarrollamos un sistema de mantenimiento predictivo que usa IA para detectar y predecir problemas antes de que ocurran. El sistema funciona bien: detecta anomalías, predice problemas futuros y recomienda qué hacer para solucionarlos.

Lo más importante que logramos:

- Creamos un sistema modular que se puede expandir
- Mejoramos las predicciones para que sean más estables
- Desarrollamos un sistema de reglas para recomendar soluciones
- Probamos que todo funciona correctamente

Para el futuro queremos:

- Usar redes neuronales más avanzadas
- Que el sistema aprenda de sus propias acciones
- Explicar mejor por qué el sistema toma ciertas decisiones
- Probar el sistema en servidores reales en producción

En conclusión, este sistema puede ayudar a reducir problemas en servicios web al predecirlos antes de que ocurran. Es un buen punto de partida para sistemas más avanzados en el futuro.

REFERENCIAS

[1] X. Zhao, J. Lin, and Z. Liu, "Deep learning-based fault prediction for industrial processes,"*IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2126-2135, 2019.

[2] C. Zhang and Y. Wang, "Anomaly detection in industrial time series data using unsupervised deep learning approaches,"*IEEE International Conference on Prognostics and Health Management*, pp. 1-8, 2018.

[3] Netflix, "Hystrix: Latency and fault tolerance library designed to isolate points of access to remote systems,"GitHub repository, 2018. [Online]. Available: https://github.com/Netflix/Hystrix

[4] F. Pedregosa et al., "Scikit-learn: Machine learning in Python,"*Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, 2nd ed. OTexts, 2018.

[6] R. K. Gunupudi and S. Tata, "A comprehensive survey on predictive maintenance for industrial internet of things,"*IEEE Access*, vol. 8, pp. 125459-125476, 2020.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory,"*Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[8] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data,"*PLOS ONE*, vol. 11, no. 4, p. e0152173, 2016.