HACKATON ZMP - Equipo Glitcheados - Pablo, Germán y Álvaro

Ideación

Objetivo

Cada día se generan grandes cantidades de piezas defectuosas en la línea de producción de motores de Martinrea Honsel Spain. Las burbujas que en ocasiones se crean dentro de las piezas basadas en aluminio pueden dar lugar a canales por donde fluye el agua desde el circuito de refrigeración al de aceite, dañando así el vehículo.

La herramienta software creada por nuestro equipo tiene por objetivo detectar dichas imperfecciones mediante una IA, para posteriormente descartar tales piezas de la línea de producción.

Solución propuesta

El equipo pretende crear un programa en el cual, usando Machine learning, y entrenando a esta aplicación con una base de datos ya conocida, sea capaz de descartar posteriormente las piezas que puedan contener burbujas.

Para su descarte, los usuarios introducirán la fiabilidad en la que quiera que estas piezas no contengan burbujas. Las piezas serán analizadas con sensores en el paso previo a su uso para la fabricación de motores.

La aplicación será previamente “entrenada” por nuestro equipo usando datos obtenidos directamente de la cadena de producción, medidos con sensores sumamente precisos.

***Las instrucciones y bibliotecas para ejecutar todo se encuentran en README.md**

Documentación de los casos de usos y requisitos

Caso	Actor	Descripción	Relación con otro caso
Establecer umbral	Trabajador	Un trabajador pretende ajustar la línea de producción para que ésta deseche automáticamente las piezas que puedan contener fallas. Para ello establece un umbral de 0 a 1	

Examinar pieza	-	El sistema examina una pieza en base a un umbral y considera si la desecha	Establecer umbral
Desechar pieza	-	El sistema conduce una pieza por otra línea de producción para desecharla	Examinar pieza

REQUISITOS:

Establecer umbral
1. Solicitar umbral 2. Comprobar que el umbral se encuentre en el rango [0,1]
Examinar piezas
1. Datos por pieza 2. Calcular fiabilidad 3. Comparar fiabilidad con el umbral 4. Devolver o no la pieza conforme la comparación anterior
Desechar piezas

Datathon

EDA y procesamiento de datos

Se realizó una primera limpieza eliminando las columnas que tuvieran todos los valores repetidos, ya que estos no aportan ningún tipo de diferencia significativa. Por otro lado, se eliminaron las filas faltos de algún elemento después de la eliminación previa, ya que se consideró que la base, con una muestra más que generosa de más de 40 mil elementos, podía permitirse la eliminación de los mismos.

El resultado de nuestra limpieza redujo considerablemente los atributos a tener en cuenta por nuestra IA.

Debido a la falta de tiempo en este proyecto, el equipo no tuvo tiempo de pulir mejor los datos dados, pero un análisis exploratorio de datos más profundo nos hubiera ayudado a eliminar los datos más redundantes, acotando de manera efectiva el problema. Se pretendió, a partir de la matriz de correlación obtenida, observar los valores y eliminar toda columna que no infiriera significativamente al atributo de fiabilidad. Para ello queríamos guardar en una lista todos los atributos cuyo coeficiente de correlación fuera mayor que 0 y a partir de la misma, usar un código similar al siguiente:

```
final_cols = [col for col in df.columns.values if col in x]
df = df[final_cols]
```

Otra forma que se nos ocurrió de pulir los datos era, similar a la anterior, guardar en una lista todos los atributos cuyo coeficiente de correlación fuera mayor que -0.2 y menor que 0.2, y eliminar así las correlaciones débiles o nulas. Esta última nos gustaba más (por no decir que descartamos la anterior idea por esta).

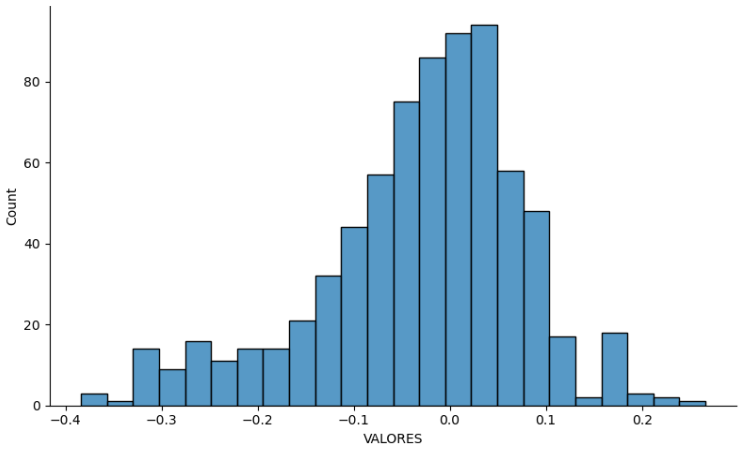

Figura 1. Valores observables de la correlación de todos los atributos con el atributo fiabilidad (si no tienen no aparecen)

Para la realización de esta parte se han empleado los siguientes métodos:

- limpieza.py: elimina las columnas cuyos valores son todos iguales y las filas que estén faltos, mínimo, de 1 elemento(no se consiguió mejorar como teníamos previsto)

En paralelo se usó la siguiente línea de comandos para sacar los valores de la correlación:

```
pd.DataFrame([[c, df['reliability'].corr(df[c])] for c in df.columns if c!='reliability'])
```

y se guardaron los datos extraídos en un csv llamado “correlation_with_reliability.csv”

Comparativa de modelos y entrenamiento del mejor modelo predictivo + R2 de las predictivas realizadas

Para hacer la comparativa de modelos, hemos usado “batch_data_train.csv” e implementado un test_entrenamiento.py.

Hemos elegido e implementado 3 modelos: lineal, de árbol de decisión y random forest. Antes de ninguna prueba, partimos de la idea de que el mejor modelo va a ser el random forest, porque hace una media de las predicciones de cada uno de los árboles, lo cual le da mucho más poder predictivo que 1 solo arbol y de base funciona muy bien con los parámetros que ya tiene (aunque es muy lento).

En la siguiente tabla se muestran los resultados de la ejecución de “batch_data_train.csv” para los modelos anteriormente comentados

Modelos	R2 por …	
	Resustitución	Partición(entrenamiento 80%, test 20%)
Linear regression	0.6442576566213256	-3.1123697505813267e+22
Decision tree regressor	**0.9996334113734923**	0.7558202142329342
Random forest regressor	0.9852645195990899	**0.8795792825753445**

Que el R2 de la regresión lineal con partición de un número negativo nos da que pensar 2 cosas: o que la realización fue la incorrecta o que es el peor modelo a elegir, pues implica que se ajusta peor que si se ajusta una línea recta a través de la media de la muestra observada (es mejor adivinar la media de los valores agrupados que utilizar la línea de regresión).

El mejor modelo a nuestro ver, una vez hechas las pruebas, es el de Random forest regressor, ya que saca el mejor R2 por partición y a pesar de que sea peor por resustitución que el Decision tree regressor, no solo la diferencia es mínima, sino que también buscamos el máximo por partición debido a que es más realista a pesar de que se desaproveche un 20% de los datos.(No obstante, en el estado final del proyecto, nos hemos quedado finalmente con el modelo “Decision tree regressor”, no porque sea mejor, sino porque el random forest puede llegar a tardar bastantes minutos)

Para la realización de esta parte se han empleado los siguientes métodos:

- entrenamiento1.py: declara la función que entrena al modelo con sustitución o con partición y genera el pickle. Dentro tenemos los 3 modelos que hemos empleado
- test_entrenamiento.py: llama a la función Entrenamiento1.
- prediccion.py: define la función que realiza predicciones
- test_prediccion.py: realiza predicciones a partir del conjunto de muestras de entrenamiento
- dividir_dataset.py: divide los datos en 2 conjuntos de muestras, 1 para su entrenamiento que supone el 80% y otra para hacer test que supone el 20% . Además ejecuta la limpieza (limpieza.py)

ZDMP Hackathon

Encapsulamiento del modelo + Despliegue con AIAR

Hemos comprimido y enviado los archivos pertinentes referentes al ML, al cluster, para el etiquetado de la fiabilidad. Para ello se han encapsulado en un .zip el fichero manfiest.json y la carpeta src con predict.py y model1.pkl, archivo resultante de realizar el entrenamiento con el mejor modelo en el punto anterior (esto se corresponde al código fuente y al manifiesto de configuración):

Al subir el .zip al servidor,este se encarga de generar una imagen DOCKER y desplegarla en un cluster Kubernetes. La imagen atenderá periódicamente mensajes con muestras y generará salidas con la predicción. Con MQTT Explorer, podemos ver los mensajes que envía y recibe nuestro servicio una vez desplegado

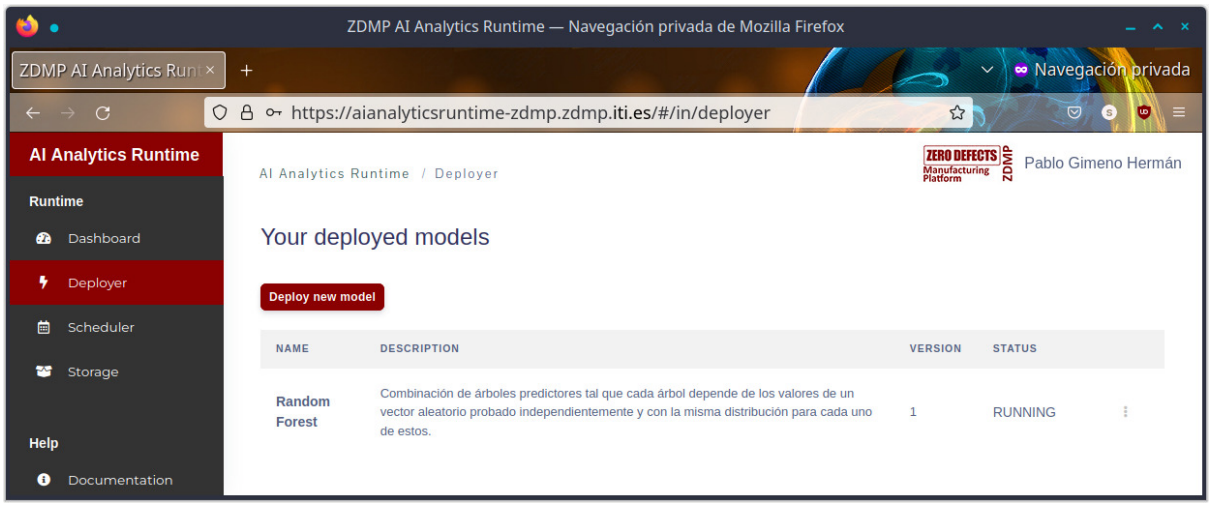

Figura 2. Despliegue en AIAR del modelo

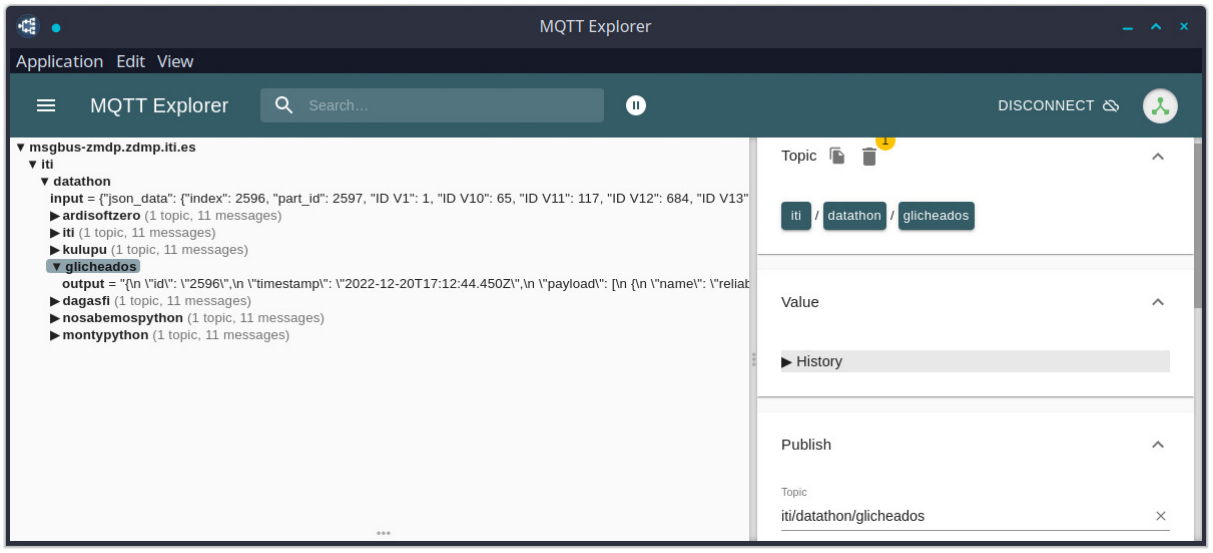

Figura 3. MQTT Explorer para observar el envío de mensajes

Implementación de la App

Por un lado hemos implementado un emisor que genera mensajes que contienen la información asociada a muestras observadas, simulando un componente conectado a los sensores que habría en el caso real.

Por otra parte, tenemos la aplicación principal con una interfaz de usuario que es capaz de leer las muestras y efectuar predicciones con el modelo entrenado, mostrando en pantalla la fiabilidad calculada y determinando si la pieza es Defectuosa o Admisible en base al umbral establecido por el propio usuario (por defecto 0.5 al iniciar la app).

Por simplicidad se ha optado por usar un fichero .JSON para la escritura de la muestra por parte del emisor y la posterior lectura por parte de la aplicación de usuario.

Cosas a tener en cuenta de la App:

- No avisa de los errores: sólo admite números en el rango de 0 a 1 para establecer el umbral, cualquier otro número o carácter no hará nada. Esta restricción está descrita en la App pero no manejamos los errores.

En un futuro se podría mejorar la eficiencia del programa automatizando la función de “Actualizar”, que hacemos manualmente, de manera que en tiempo real avisará de las piezas defectuosas e implementarlo mediante sockets para separar los componentes en la red

Figura 4. Interfaz final de la aplicación