Predicción de los fallos de fabricación en la línea de producción

Eduard López García

Resumen— A día de hoy las líneas de producción más modernas disponen de centenares, sino miles de sensores por los que pasan las piezas y productos para ser fabricados. Este hecho abre un abanico de posibilidades a la hora de extraer datos útiles de dichos sensores. Uno de sus usos principales, es decidir si un producto pasará o no el control de calidad a partir del análisis de los sensores por donde ha pasado dicho producto. En este trabajo se intenta dar solución a este problema. Los datasets a usar han sido proporcionados por la empresa Bosch, datasets que hace un tiempo hizo públicos [1]. Este trabajo empieza con un análisis de los datasets, y dos propuestas prácticamente opuestas de resolución de este problema. Este documento bien podría llamarse: "Técnicas de cómo no hacer un modelo de predicción de fallos de fabricación en la línea de producción". Dado que se exponen los métodos y aproximaciones al problema que han fallado y porqué. También se expondrán las técnicas que han surgido efecto y el porqué de ellas. Y por acabar se expondrán las propuestas de mejoras futuras, que no se han implementado por falta de tiempo, con el objetivo de mejorar más en la predicción.

Palabras clave— Fallos de fabricación, fallos línea de producción, aprendizaje computacional en la fabricación, línea de producción XGBoost.

Abstract— Nowadays the most modern production lines have hundreds, if not thousands of sensors where pieces and products to be made go through them. This fact opens a big range of possibilities at the moment of extracting useful information of those sensors. One of their principal uses is to decide if a product will pass or not the quality control from the analyses of the sensors through each product has passed. The solution of this problem is tried to be given in this work. The datasets used have been provided by the company Bosch, datasets that were done public a time ago. This work starts with an analysis of the datasets, and two practically opposite offers of the problem resolution. This document could also be called: "Techniques of how not to make a prediction model of manufacturing failures in the production line". Due that the methods and approaches of the problem that have failed and why are exposed. It will also expose the techniques that have worked and the reason why. And lastly the proposals of future improvements, which haven't been implemented due to lack of time, with the objective of improving the prediction more, will be exposed.

Index Terms— Manufacturing failures, production line failures, machine learning in manufacturing, XGBoost production line.

1 Introducción Y Objetivos

Gracias al avance tecnológico podemos disfrutar de productos con funcionalidades nunca antes vistas. Pero los procesos de fabricación de estos requieren cada vez más de métodos más técnicos y avanzados.

Los procesos de fabricación masivos [2], también llamados en cadena o en serie, son esenciales para satisfacer las necesidades de una población creciente [3] por lo que siempre se están buscando nuevos y mejores métodos para mejorar que los productos que salen de una línea de producción [4] cumplan los requisitos que el fabricante ha establecido y que este funcionará tal y como fue diseñado.

Detectar anomalías en los procesos de producción que pueden llegar a causar que el producto terminado falle al ser enviado al comprador, es un tema de vital importancia para el fabricante, dado que entregar un producto mal fabricado puede causarle desde mala reputación hasta problemas más graves que pueden poner en riesgo la vida humana. Además de que los fallos incrementan los costes y acaba repercutiendo en productos más caros. Es por ello que los fabricantes están continuamente probando nuevos métodos para detectar si un producto fallará o no al salir de la línea de producción

Hoy en día se pueden disponer cada vez de más y baratos sensores que se pueden repartir a lo largo de las líneas de producción para recopilar datos sobre el producto a medida que pasa por el proceso de fabricación para luego determinar si pasará o no el control de calidad [5] [6]

Dado que la idea de la producción masiva es que se produzca la mayor cantidad de productos en el menor tiempo posible a una calidad deseada, se requiere que en cada fase incluyendo el control de calidad sea del menor coste en tiempo posible.

Por lo que se requiere de un modelo que a partir de los datos recogidos a través de la línea de producción sea capaz de predecir si el producto pasa o no el control de calidad, es decir, si el producto es defectuoso o no.

Explicado el contexto, este Trabajo Final de Grado (TFG)

- *E-mail de contacte: eduard.lopezg@e-campus.uab.cat*
- *Menció realitzada: Computació*
- *Treball tutoritzat per: Dr. Ramon Baldrich Caselles (Departament de Ciències de la Computació)*
- *Curs 2016/17*

consistirá en lo explicado anteriormente, en generar un sistema que diga si un producto pasa el control de calidad o no analizando los *n* sensores por los que pasa a través de la línea de producción.
La técnica necesaria para la realización de tal propósito será un área de las ciencias de la computación llamada Machine Learning [7] [8] (también conocida como Aprendizaje Automático o Aprendizaje Computacional entre otros). Machine Learning es una rama de la Inteligencia Artificial, que explicada de forma resumida trata la manera de que a partir de unos datos de entrenamiento *dataset de training* (ej.: datos históricos de productos que han pasado por los sensores y que sabemos si han pasado satisfactoriamente el control de calidad o no), sea capaz de crear un modelo que a partir de unos datos nunca vistos, datos de *test*, (ej.: datos como los de *training* pero que el modelo no haya visto aún) pueda producir una respuesta o *label* (ej.: si el producto ha pasado el control de calidad o no).

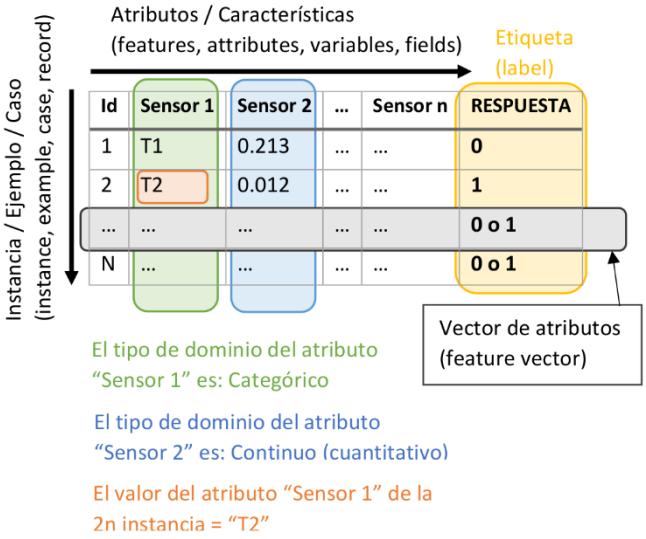


Ilustración 1 Definición de cada termino dentro de un dataset [9] [10] Un dataset puede presentarse de muchas formas. Un conjunto de imágenes o sonidos puede ser un dataset. En nuestro caso es un dataset en forma de tabla.

Los objetivos principales del proyecto son los siguientes:

- **Fase de Investigación:**
El objetivo principal es el de realizar una investigación del problema a tratar para determinar los pasos a seguir a continuación.
En esta fase se determinará el modelo a ejecutar a continuación como también el tratamiento que tiene que sufrir el Dataset para que se le pueda aplicar el modelo. Dado que, dependiendo del modelo, necesitará una presentación de los datos de entrada de una forma u otra. También se determinarán las tecnologías a utilizar.

- **Preparación del Dataset**
Según lo que haya sido decidido en la fase de investigación se aplicarán las transformaciones necesarias al dataset para que pueda ser tratado en la fase de ejecución del modelo 1 de a continuación.

- **Modelo 1**
Aquí se aplicará un modelo elegido en la fase de investigación.

- **Análisis siguiente modelo**
Se analizarán los resultados obtenidos en la ejecución del modelo 1 para determinar las modificaciones necesarias para ejecución del modelo siguiente.

- **Modelo 2**
Como objetivo final, el fin es aplicar otro modelo completamente distinto al 1r además de tener en cuenta lo que ha salido mal del 1r para mejorar el 2n.

2 ESTADO DEL ARTE

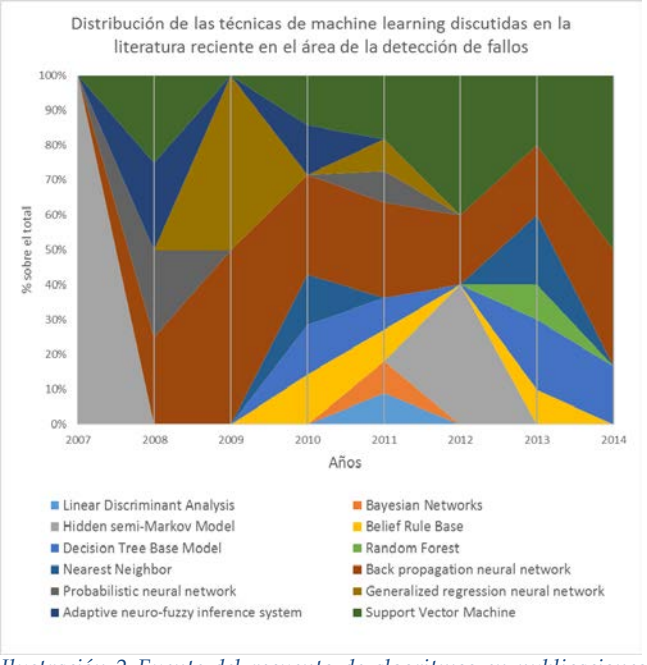


Ilustración 2 Fuente del recuento de algoritmos en publicaciones: [11] Datos completos en el Anexo A3

Este trabajo trata de solucionar los problemas de fallo de fabricación con un dataset proporcionado por la Bosch en una planta inmensa, con muchas líneas de producción, estaciones y sensores. Por lo tanto, este trabajo trata de solucionar un problema de una fábrica grande.
No hay un método único por donde atacar el problema. Grandes empresas venden software que se encasta directamente con la línea de producción para detectar fallos, pero evidentemente estas no hacen público como funcionan sus modelos al detalle.
Aún menos cuando hay una gran cantidad de datos, donde se tienen que adaptar técnicas para ser capaces de atacar los datos.
Por lo que hay publicado sobre el tema, es común el uso de distintos tipos de árboles de decisión. Y los más sofisti-

cados con Redes Neuronales como las Recurrentes dado que son buenas en el ámbito de las series de datos. A más a más se usan métodos *ensembler* con los algoritmos anteriores u otros de los que no se sabe.
En la Ilustración 2, podemos ver de forma aproximada, sobre la literatura que hay publicada sobre el tema, los algoritmos usados para el propósito de la detección de fallos. Los datos del sondeo han sido extraídos de: [11]
Como podremos observar, el modelo principal que se usará en este trabajo, no está entre los más populares, motivo de más para explorar las posibilidades de este.

3 Metodología y Planificación

Se ha elegido un método de desarrollo en cascada donde el enfoque metodológico es el que cada etapa debe esperar a la finalización de la etapa anterior [12]
Se ha dividido el proyecto en dos áreas principales: la del desarrollo de éste propiamente dicho y el área de comunicación.
En la Tabla 1 puede verse el resumen de las tareas en el área del desarrollo del proyecto.
Y en la Tabla 2 las tareas en el área de la comunicación del desarrollo del proyecto. Tareas que tienen como fin el comunicar el progreso de las tareas de la Tabla 1.
Cada tarea tiene asociado un tiempo de duración estimado y las fechas, también estimadas, de cuando se tiene programado dar comienzo y fin a dichas tareas.
Con el objetivo de un mejor entendimiento de estas, se han proyectado las tareas de las 2 áreas en un diagrama de Gantt (ver apéndice A1), esto facilita mucho el entendimiento y orden temporal de las tareas. También se puede visualizar la aplicación de la metodología en cascada sobre las tareas a realizar.

Nombre de tarea	Duración	Comienzo	Fin
DESARROLLO DEL PROYECTO	**100 días**	**lun 12/09/16**	**vie 27/01/17**
Fase de Investigación	**38 días**	**lun 12/09/16**	**mié 02/11/16**
Análisis del Dataset	10 días	lun 12/09/16	vie 23/09/16
Investigación sobre cuestiones teóricas	14 días	lun 26/09/16	jue 13/10/16
Investigación sobre las tecnologías a utilizar	14 días	vie 14/10/16	mié 02/11/16
Preparación del Dataset	10 días	jue 03/11/16	mié 16/11/16
Ejecución nº1	**25 días**	**jue 17/11/16**	**mié 21/12/16**
Modelado Y Evaluación	25 días	jue 17/11/16	mié 21/12/16
Análisis siguiente modelo	7 días	jue 22/12/16	vie 30/12/16
Ejecución nº2	**10 días**	**lun 02/01/17**	**vie 13/01/17**
Modelado Y Evaluación	10 días	lun 02/01/17	vie 13/01/17
Borrador del artículo	5 días	lun 16/01/17	vie 20/01/17
Acondicionamiento del código para entregar	5 días	lun 23/01/17	vie 27/01/17

Tabla 1 Planificación temporal en el área del desarrollo del proyecto

Nombre de tarea	Duración	Comienzo	Fin
ÁREA DE LA COMUNICACIÓN	**114 días**	**lun 12/09/16**	**jue 16/02/17**
REUNIÓN INICIAL	5 días	lun 12/09/16	vie 16/09/16
ENTREGA - Informe Inicial	6 días	lun 26/09/16	dom 02/10/16
ENTREGA - Informe 1	6 días	lun 31/10/16	dom 06/11/16
ENTREGA - Informe 2	6 días	lun 12/12/16	dom 18/12/16
ENTREGA - Propuesta Informe Final	6 días	lun 16/01/17	dom 22/01/17
ENTREGA - Propuesta Presentación	6 días	lun 30/01/17	dom 05/02/17
ENTREGA FINAL	1 día	mar 07/02/17	mar 07/02/17
ENTREGA POSTER	6 días	lun 06/02/17	dom 12/02/17
DEFENSA TFG	4 días	lun 13/02/17	jue 16/02/17

Tabla 2 Planificación temporal en el área de la comunicación del desarrollo del proyecto:

4 Resultados de las tareas planificadas

4.1 Fase de investigación

4.1.1 Análisis del Dataset

Analizar con que datos estamos trabajando es la primera tarea a realizar dado que si no sabemos con qué tipo de datos estamos trabajando, no podremos tomar una decisión sobre qué modelo de aprendizaje y algoritmos elegir ni con qué tecnologías trabajar.
Cabe decir que las fases previas de colectar los datos en RAW, preprocesado y limpiado ya han sido realizadas por los proporcionantes de los datos. Aunque parece que los datos estén listos para generar el modelo, no es así, dado que hay que adaptar los datos a la entrada del modelo.
El paso del análisis del Dataset, también es conocido como EDA (Exploratory Data Analysis) o Análisis exploratorio de datos, es decir, realizar una exploración y análisis de los datos para tener una idea sobre qué tipo de datos estamos trabajando para así determinar cómo trabajar con ellos en las siguientes etapas.
Tenemos 2 Datasets principales, uno de training y otro de test. Al ser muy grandes cada Dataset está partido en 3 partes, ej. *train*: train_categorical.csv, train_numeric.csv, train_data.csv

Dataset train: (el dataset test es similar):
Contiene 1 183 743 filas. También llamadas observaciones.
1 176 868 están etiquetas como 'Response' = 0 (0 representa que el producto si ha pasado el control de calidad)
6 879 están etiquetas como 'Response' = 1 (1 representa que el producto no ha pasado el control de calidad)
Es decir: una ratio de 1:172
Aquí podemos observar lo poco balanceados que están los datos. Esto puede provocar un problema a la hora de generar el modelo, dado que al haber tan pocas de un tipo ('Response' = 1), en general, tenderá a predecir solo de un tipo. Es un punto que se habrá de tratar en la fase de Investigación sobre cuestiones teóricas.
Si se predicen todos los datos como negativos (los que hay mas), estaríamos hablando de aproximadamente 99.4% de acierto

Contiene 4 262 columnas correspondientes a las mediciones de los sensores. También llamadas features o características.
De estas 2142 son categóricas, 971 numéricas y 1556 *timestamps* o fechas de cuando fue tomada cada medida.
En RAW, o en bruto, todo el Dataset (las 3 partes) tiene un tamaño de aproximadamente 14.3GB
Además, hay 3 líneas de producción y un mismo producto no tiene por qué empezar y terminar en la misma línea de producción. Una exploración del dataset indica que hay alrededor de unos 7000 caminos únicos por donde pueden pasar los productos.

4.1.2 Investigación sobre cuestiones teóricas

En esta fase, se ha realizado una investigación sobre el tema del que trata el reto de este trabajo (el de predicción de fallos en la línea de producción).

El principal factor de análisis ha sido qué algoritmos hay disponibles y qué se pueden aplicar a dicho problema, teniendo en cuenta que este se trata de un problema de aprendizaje supervisado [13] y que hay que escoger un algoritmo acorde al problema.
Al analizar los distintos tipos de algoritmos, los árboles de decisión parecen una buena opción. Analizando los tipos disponibles de árboles de decisión, nos encontramos con el modelo de (*boosted trees* [14]) expuesto en: *Greedy Function Approximation: A Gradient Boosting Machine, by Friedman* [15]. La implementación más famosa es la llamada XGBoost [16]. Dicho algoritmo está descrito con detalle en el apartado 4.1.2.1

Se han elegido 2 enfoques de modelado con el fin de solucionar el problema:

Modelo 1
En este modelo se ha enfocado a usar contra más datos mejor sin hacer ninguna selección de características (reducción de dimensionalidad).
En este modelo se tratará el problema del inbalanceo de datos. Tenemos muchos de una clase y pocos de otra. Por lo que es necesario un método para solucionar este problema. Utilizaremos *Over-sampling* con la técnica de generación de datos sintéticos: SMOOTE [17] (ver apartado 4.1.2.1) con el objetico de generar las muestras necesarias del tipo menos representado para igualarlas 1:1
Con ello quedará un dataset gigantesco, por lo que es necesario de un equipo lo suficientemente capaz para poder tratarlo (ver más en el apartado del Modelo 1)

Modelo 2
Para el 2n modelo se seguirá una aproximación más "inteligente"
Se analizarán las características que aporten más información, determinaremos las *n* características más importantes.
Entonces procederemos a hacer un *subsampling*, en concreto del tipo *Stratified Sampling*. Con ello conseguiremos un *dataset* muestreado con todas las categorías representadas de forma proporcional, cosa que no podríamos asegurar con un *subsampling* [18] aleatorio, dado que hay muchas categorías y muchas de ellas poco representadas.
Con todo ello conseguiremos un dataset muestreado más pequeño y manejable tanto en número de categorías como en número de muestras.

4.1.2.1 SMOTE

En los gráficos de dispersión de la Ilustración 3 podemos observar 2 clases de datos coloreados en color verde y magenta. Como se puede observar disponemos de menos datos de color magenta, lo que puede causar problemas a la hora de generar el modelo por los motivos expuestos con anterioridad. En este paso la solución es generar datos sintéticos, que si nos ceñimos a la definición de la *McGraw-Hill Dictionary of Scientific and Technical Terms*, sería la "cualquier producción de datos aplicable a una situación en que los datos no se obtengan por una medición directa". En este caso estaríamos generando datos sintéticos a partir de datos que sí han sido tomados de forma directa. Un buen algoritmo que puede solucionar el problema es el llamado SMOTE. De forma general este algoritmo funciona generando nuevas instancias a partir de ya existentes a partir de la combinación de características de las de sus vecinos, esto hace que las muestras generadas sean más generales.

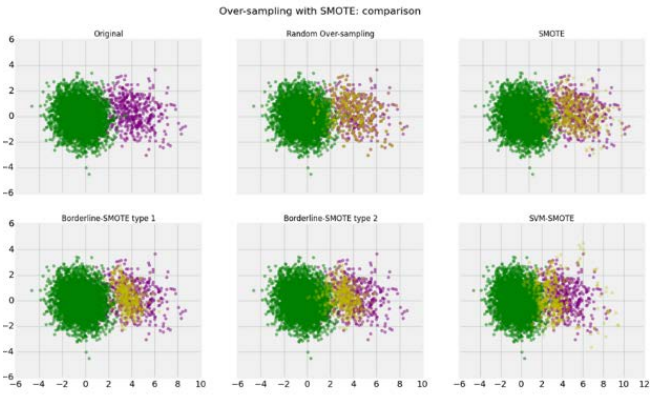


Ilustración 3

4.1.2.1 XGBoost

XGBoost es la abreviación de *Extreme Gradient Boosting*.
Gradient Boosting es una técnica en el área del *Machine Learning* usada principalmente en el área de clasificación y la regresión.
La técnica principal de la que deriva el algoritmo es la de un tipo de *Ensemble Method,* el *Boosting*.
El *Ensemble Method* [19] es una técnica en la que muchos modelos predictivos sencillos o *predictors* son unidos para formar un único modelo más complejo. La importancia de cada uno de los *predictors* sencillos dentro del clasificador final viene dada por una métrica (un peso, votos, etc.) que cada uno tiene.
Como se ha dicho anteriormente, el *Boosting* es una técnica *Ensemble*, en la que una combinación de *weak learners* crean un *strong learner* o *complex predictor*. Y eso se consigue de forma iterativa, donde el siguiente *weak learner*, aprende de los errores del anterior *learner*.
El *Gradient Boosting* [19] recoge las ideas de las técnicas anteriores de un modo particular. Como mejor se entiende es con un ejemplo como el de a continuación.

Ejemplo de *Gradient Boosting* (con un ejemplo de regresión):

1) A la izquierda (Ilustración 4) tenemos un *dataset* (puntos rojos) donde se le aplica un *predictor* sencillo (línea negra), en este caso un árbol de decisión de una sola capa partiendo el espacio en 2 partes determinado por un umbral.
2) Calculamos el error residual generado por el *predictor* de la izquierda (Ilustración 4). Los errores se muestran en la imagen de la derecha (Ilustración 4)
3) Ahora procedemos a generar a otro *predictor* (lí-

nea negra) (Ilustración 4), a partir de los errores generaros (imagen de la derecha) (Ilustración 4)

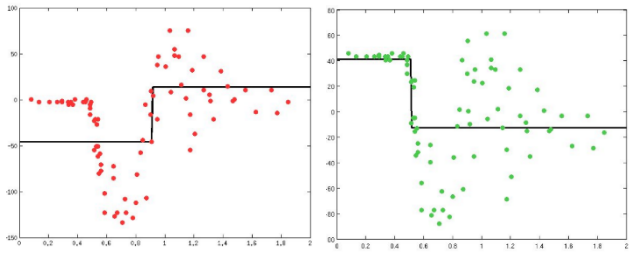


Ilustración 4

4) Al sumar los 2 *predictors* (de la Ilustración 4), se genera un predictor más complejo (imagen de la izquierda) (Ilustración 5). Ahora tenemos una función con 2 puntos de transición y sin lugar a duda se ajusta mejor a los datos que no un solo *predictor* árbol de una sola capa.
5) Ahora procedemos a calcular los errores generares por el nuevo predictor (imagen de la derecha) (Ilustración 5) y calculamos un nuevo *predictor* a partir de los nuevos errores.
6) Repetimos todo el proceso

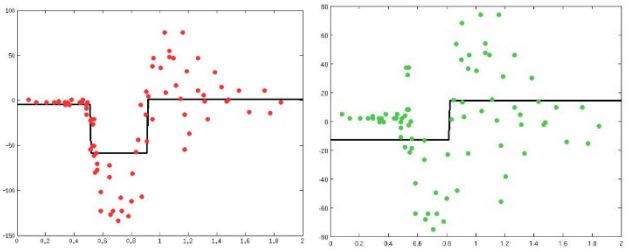


Ilustración 5

En las imágenes de la Ilustración 6 podemos observar más iteraciones del algoritmo con sus *predictors* y sus correspondientes errores residuales necesarios para calcular los siguientes.

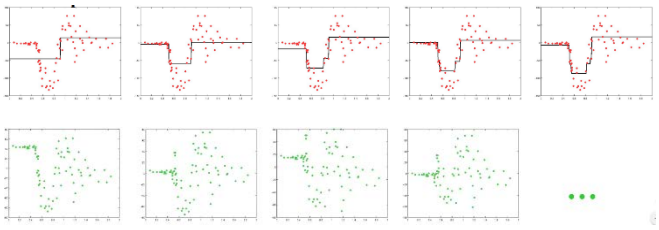


Ilustración 6

Como podemos observar, no estamos atados a usar un solo tipo de *predictors*, podríamos usar muchos modelos diferentes a la hora de ajustar los datos a la función del modelo escogido. El modelo escogido en XGBoost son arboles de decisión. En particular el llamado *Tree Ensembles*. El modelo Tree Ensembles es un conjunto de *Classification and Regresion Trees* (CART).

Por último, el que XGBoost se llame *eXtreme* es debido a que fue concebido por sus creadores como una herramienta enfocada en la optimización y la ejecución en paralelo.

4.1.3 Investigación sobre las tecnologías a utilizar

Sistema Operativo (SO):

- El SO que se ha usado para la realización de este trabajo es Linux. En particular la distribución Ubuntu.
- Se ha elegido Linux dado que después de analizar múltiples herramientas, lo que tenían en común era que eran para Linux o iban mejor sobre Linux, por lo que al final la mayoría manda y Linux es la opción elegida.

Lenguajes:

- El lenguaje principal será Python, dado los innumerables recursos que hay ya creados bajo este lenguaje. Y no es solo el lenguaje sino todo el ecosistema de herramientas, librerías y recursos sobre IA, *machine learning*, estadística, tratamiento de datos, etc. Que hay formado entorno al lenguaje.
- Otras herramientas que se han usado ha sido el bash de Linux dado que comandos como head, tail, cat, y combinaciones de ellos en pipelines son de alta utilidad para el tratamiento de ficheros de datos.

Entorno de desarrollo / herramientas / librerías:

- A la hora de crear el entorno de desarrollo hay varios puntos a tratar. La versión del lenguaje, las librerías, las dependencias de las librerías, el que haya librerías que sean incompatibles con la versión del lenguaje que utilizas, etc.
  Esto hace que haya optado por usar la Anaconda [20]. Anaconda es un sistema de gestión de desarrollo que permite una ágil gestión de los paquetes necesarios para el funcionamiento del proyecto.
  Lo consigue creando distintos entornos virtuales. Por ejemplo, se puede crear el entorno llamado "tfg" y cuando se activa, todas las llamadas que se hagan al lenguaje Python y librerías se hará dentro de este entorno. Por lo que si por ejemplo ejecuto un .py, el sistema no llamará al Python instalado en el SO, sino el que hay instalado en el directorio del entorno virtual "tfg". Esto permite que se puedan crear distintos entornos de trabajo con distintas versiones del lenguaje y con distintas versiones de librerías sin que unas interfieran con las otras.
- Jupyter Notebooks, también conocidas como "libretas de programación" donde se ha procurado ir escribiendo el código de forma que se pueda leer como un libro, hecho que tiene como objetivo una mejor organización y entendimiento el código.
- Librerías de machine learning: la principal scikit-learn. Es una librería con una cantidad inmensa de algoritmos ya implementados, es la librería rey en su ámbito.
- Ecosistema SciPy (NumPy, Matplotlib, IPython, pandas, etc.) [21] En particular NumPy se utiliza en tan

gran cantidad de proyectos que es casi imposible no utilizarlo. Es una librería que, a rasgos fuertes, pretender el poder utilizar estructuras de arrays de forma mucho más eficiente que las implementadas nativamente en Python.

- Como se ha comentado anteriormente, para generar el modelo 1, se utilizará la implementación XGBoost escrita en Python.

4.2 Modelo 1

Tal y como se ha comentado anteriormente, el enfoque de este primer modelo es el de usar contra más datos mejor sin realizar ninguna reducción de dimensionalidad (a parte de las timestamps por los motivos expuestos más abajo)

La idea subyacente es que el algoritmo aprenda con los datos más en bruto posible que salen directamente de los sensores de la línea de producción sin importar que tipo de sensores sean. Aunque pueda parecer una ventaja, eso puede inducir a un aumento de ruido en el modelo dado que no hacemos ningún tipo de criba ni trato en ninguna de las características del *dataset*.

También se decidió que el modelo fuera atemporal. Es decir, no se tendrían en cuenta las fechas en que las medidas de los sensores fueron tomadas.

Otro tema principal es el del trato del imbalanceo de las muestras. En el dataset de train tenemos 1 176 868 muestras con el *label* 'Response' = 0 y 6 879 con el *label* 'Response' = 0. Es decir, una ratio 1:172. Para solucionar este imbalanceo con ratio, procederemos a usar una técnica de generación de muestras sintéticas SMOTE (ver apartado 4.1.2.1) con el objetivo de igualar la ratio a 1:1.

4.2.1 Hilo de ejecución

Tal como se muestra en la figura del Apéndice A2, se puede observar 2 principales flujos de ejecución, la local y la realizada en un servidor. Pero las 2 comparten el mismo hilo de ejecución.

La idea es subdividir todo el problema en ficheros de código separados. Cada fichero de código tendrá unos datos de entrada pertenecientes a la salida producida por el fichero de código anterior. Las ventajas de subdividir el código de esta manera y que en cada paso se generen ficheros de datos intermedios es que permite una mayor reutilización de los datos, por ejemplo, si se quiere volver a computar el paso nº4, no hace falta volver a calcular el 1, 2 y 3, sino que solo hay que tomar la salida del fichero 3 que ya tenemos guardada. Aunque con muchas diferencias, se asemeja al concepto de *pipeline* [22] de los sistemas UNIX-like.

Hay 6 pasos principales:

1- Preparar el dataset: Transformación de datos categóricos a numéricos, tratamiento de los *missing values*, etc.
2- Generar un muestreo de prueba: Para ello se tomarían la salida del paso anterior y se generaría un *subsampling* aleatorio de los datasets originales
3- Unir los datasets: Aquí se procedería a unir los diferentes datasets de muestreo que se han generado en el paso anterior
4- Generar datos sintéticos: Al estar las etiquetas que indican la *label* de las observaciones (lo que hay que predecir, 'Response'=0 o 'Response'=1) altamente imbalanceadas (ratio 1:172). Se procede a generar tantas observaciones como sean necesarias para igualar la ratio a 1:1
5- Generación del modelo: Se genera el modelo XGBoost a partir de los datos anteriores.
6- Evaluación: Se evalúa el rendimiento del algoritmo del paso anterior con los datos de test. La métrica a utilizar será la *Matthews Correlation Coefficient* (MCC) [23]

4.2.1 Ejecución en local

Dado el gran tamaño de los *datasets*, se hace imposible comprobar el correcto funcionamiento del código con todos los *datasets* completos. Por lo que en esta parte se ejecutará el código con *subsampling* aleatorio de los *datasets* bastante pronunciado.

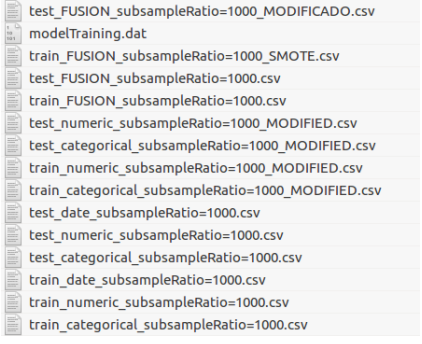


Ilustración 7. Ficheros intermedios la ejecución en local.

Se procedió a evaluar el rendimiento del modelo generado. Con una partición de *training-test* 70%-30%, el MCC daba como resultado alrededor del +0.9. Este resultado tan bueno sin embargo no se puede tomar muy en serio. Dado que al hacer un *subsampling* tan profundo, se entrenó el modelo con un dataset en el cual no todas las características (*features*) estaban representados de igual manera. Había características que bien no tenían ningún valor en ninguno de los casos y características que, aunque tuvieran algún valor en los N casos muestreados, estos no estaban proporcionalmente representados de igual en comparación con el dataset original.

Esto conllevó a entrenar un modelo muy ajustado a unos pocos datos. Lo que provocó *overfitting* [24] también llamado sobreajuste. Es decir, el modelo se entrenó para unos datos muy concretos, cosa que conlleva a que no pueda generalizar.

Pero el objetivo principal de la ejecución en local es la de comprobar que todo funcione correctamente, y proceder a ejecutar el código en un servidor en el que sea posible realizar los cálculos con el dataset completo y por lo tanto con la creencia de poder mitigar el problema comentado

anteriormente al entrenar el modelo con el *dataset* completo.
Cabe destacar que se invirtió mucho tiempo organizando y reescribiendo el código de las tareas anteriores para hacerlo todo más eficiente.

4.2.3 Ejecución en servidor

Una vez comprobado que el código funciona, se procedió a ejecutar todo el hilo de ejecución con los datasets enteros.
Para realizar los cálculos, alquilé un servidor en Google Cloud Platform.
En concreto, 1 Instancia de VM con 24vCPUs y 156GB de RAM. Al no ser suficiente los 156GB de Memoria Principal, le añadí un disco persistente SSD de 500GB y lo configuré como *swap* para que así el SO no mate la ejecución del programa por falta de memoria. Y por último se le añadió un disco de 400GB para el arranque con el SO (Ubuntu 14.04)
Cuando estuvo en funcionamiento, procedí a configurar las credenciales, el *tunneling* SSH y transferí el entorno de Anaconda que tenía en local.
Una vez que todo estuvo configurado, procedí a ejecutar los códigos.
Las partes más intensivas en memoria fueron las de generación de datos sintéticos y la de generación del modelo.
Pero la parte de generación de datos sintéticos, era muy intensiva en memoria, tanto que tuve que reescribirla para hacerla aún más eficiente.
Que los algoritmos SMOTE y XGBoost pudieran ser ejecutados en paralelo ayudó a disminuir el número el tiempo de ejecución.
Aunque se ejecutó en una maquina bastante potente (24

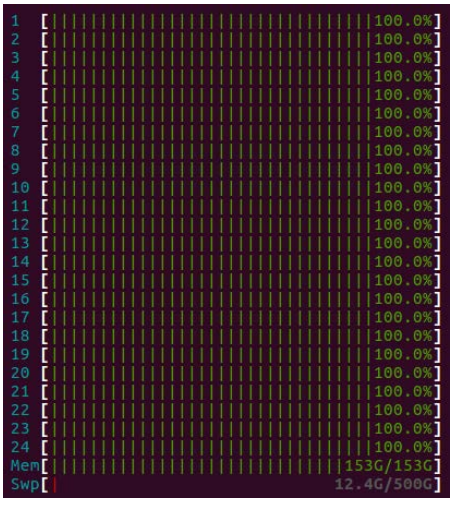


Ilustración 8. Captura de pantalla de la ejecución en remoto.

cores), el tiempo de ejecución en remoto total de una sola ejecución sin interrupciones es algo más de 6 horas.

Al final el modelo resultante no fue satisfactorio. El MCC es muy bajo. Hay varios factores que contribuyeron a este hecho.

1) Los datos sintéticos generados añadieron mucho ruido al modelo
2) La decisión de hacer el modelo atemporal. Como podrá verse en el apartado del Modelo 2, las características de fecha tienen una alta importancia dado que aportan mucha información al modelo. No es buena idea hacer el modelo atemporal
3) La eliminación de características o *features* de poca importancia. Algo que también podrá verse en el apartado del Modelo 2.
   El planteamiento del Modelo 1 era el de no realizar reducciones de dimensionalidad, se ha comprobado que es un mal camino. Han surgido problemas relacionados con la alta dimensionalidad al no haber hecho una reducción de estas (ver *Curse of dimensionality* [25])

4.3 Modelo 2

Como se comentó en la fase de investigación, el Modelo 2 se ha planteado como casi opuesto al Modelo 1.
Además, sabiendo lo que no funciona podemos hacer un modelo mejor.
Los cambios principales en este modelo son:

- Se hará una agresiva reducción de dimensionalidad
- No se generarán datos sintéticos
- Los muestreos serán estratificados

4.3.1 Identificación de los mejores atributos – Reducción de dimensionalidad

Hay muchas técnicas para determinar la importancia de las características más importantes para luego realizar una reducción de dimensionalidad eliminando las que no aportan información suficiente según la métrica de cada algoritmo en particular y el umbral que nosotros creamos adecuado que determine cuáles son importantes y cuáles no.
Para este caso he creído adecuado el usar el propio algoritmo XGBoost para ayudar a determinar la importancia de cada característica. Estos son los pasos:

1) Crear un *dataset* muy pequeño mediante un muestreo estratificado, así cada característica estará proporcionalmente representada.
2) Entrenar el modelo con el dataset anterior.
3) Extraer la importancia de cada característica. Para ello se utiliza una métrica que tiene en cuenta la mejora en el rendimiento en cada punto donde hay una bifurcación de una característica. Luego se pondera por el número de observaciones. También conocida como índice Gini

Si analizamos los 3 tipos de *features* diferentes obtenemos una distribución de importancia como la que se muestra en la Ilustración 9. Como se observa, hay muchas features que aportan muy poca información. Después deberemos elegir un umbral de corte que creamos adecuado que elimine el mayor número de *features*, tal y como se muestra en la Tabla 3.

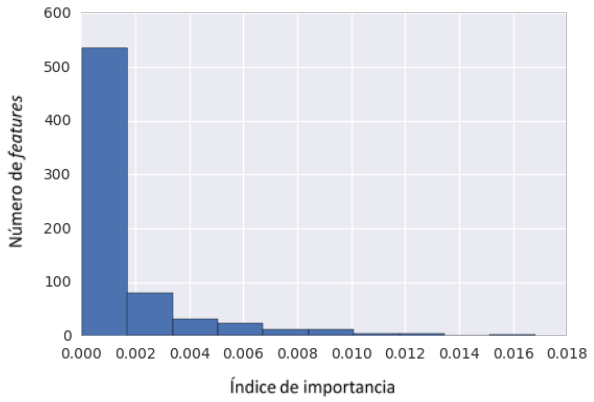


Ilustración 9 Ejemplo de histograma donde se muestra la distribución de importancias de las características.

Tipo de atributos	*En origen*	*Después de la reducción*	*%Reducción*
Tipo fecha	1156	569	50.77 %
Categóricos	2149	4	99.71 %
Numéricos	971	132	86.4 %
Total	*4669*	*705*	84.9%

Tabla 3 Ejemplo de reducción de features con un valor inferior a 0.000001

Como puede observarse, los datos categóricos aportan muy poca información. Mientras que los de fecha (timestamps) y sobretodo los numéricos, aportan gran cantidad de información.
Por lo que queda confirmado los resultados del Modelo 1; hacer un modelo atemporal es una mala idea dado que las categorías de fechas aportan gran cantidad de información.

4.3.2 Generación del modelo

Antes de proceder a generar el modelo, debemos generar un muestreo teniendo en cuenta el paso anterior.
Se reduce el dataset muestreado en 2 maneras:

1) Reduciendo el número de dimensiones (ver apartado anterior)
2) Reduciendo el número de muestras, pero sin ello perder información por el camino como pasaba con el muestreo aleatorio. En este modelo se hace un muestreo estratificado

Una vez tenemos el dataset preparado, procedemos a entrenar y generar el modelo.
Una vez entrenado el modelo, en si no nos dice de forma categórica a que clase pertenece, sino que le otorga una puntuación o *score*.
Ahora hay que determinar el umbral a partir del cual consideramos que pertenece a una clase o a otra. Como métrica para determinar la calidad de la clasificación, utilizaremos la métrica ya comentada con anterioridad, la MCC.
La manera que se ha resuelto el problema es ir prediciendo la partición de test con incrementos de 0 a 1 y luego encontrar que umbral ha hecho que el MCC sea máximo, un resultado de ejemplo es como el que se muestra en la Ilustración 10.

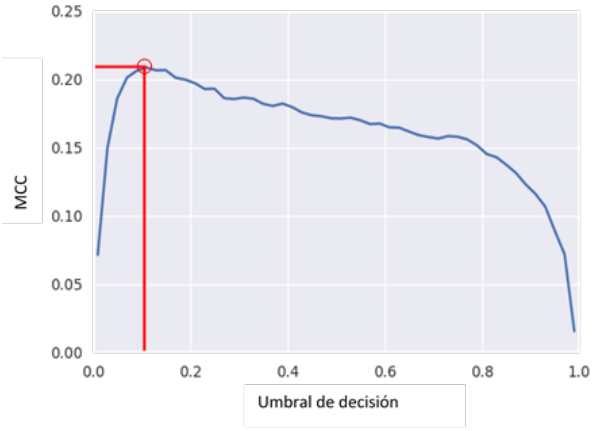


Ilustración 10. Grafico que muestra los valores de la métrica MCC vs distintos valores de umbrales de decisión.

4.3.2 Resultados

Teniendo la cuenta los nefastos resultados el Modelo 1, el Modelo 2 ha sido un éxito.
Esto confirma las premisas con las que se propuso el Modelo 2.
Después de comprobar el resultado del modelo en local con un dataset test extraído del *dataset train*, se procedió a generar el archivo de respuestas a partir del dataset *test* del que no tenemos la categoría a la que pertenece cada muestra.
Una vez subido a Kaggle, el mejor modelo dio un resultado MCC de 0.23X. Para comprarlo, el mejor resultado con un dataset test extraído del dataset de training nos dio un MCC de 0.20X. Como vemos el modelo no ha perdido precisión con unos datos con los que nunca ha sido entrenado. Por lo tanto, podemos afirmar que nuestro modelo generaliza bien y no tendrá problemas con muestras futuras.

Para terminar, se han hecho distintas comprobaciones.
La primera es la relación que hay con la cantidad de características con las que entrenamos el modelo VS el MCC que obtenemos. Como podemos observar en la Ilustración 11, hay un número de features optimo por el cual se maximiza la precisión del modelo. Por lo tanto, queda claro que la parte de extracción de *features* es de vital importancia para la generación de un modelo competente.

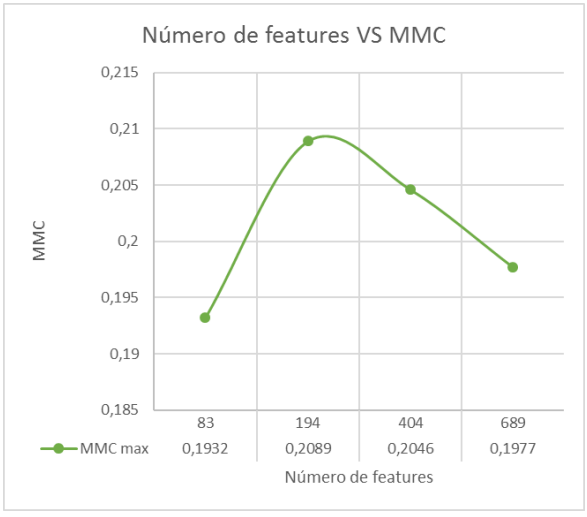


Ilustración 11. Demostración grafica de que hay un número óptimo de features con las que se maximiza la precisión del modelo

Otra comparación es la de las métricas a utilizar para medir la calidad de un modelo. El modelo ha sido evaluado con 2 métricas. La AUC (*Area Under Curve*) de la curva ROC [26]. Y la MCC. Como puede observarse en la Ilustración 4, parece que hay una relación, pero esta no es fuerte, por ello no extraigo una relación concluyente sobre si se puede establecer una relación entre estas 2 métricas aplicado a este modelo.

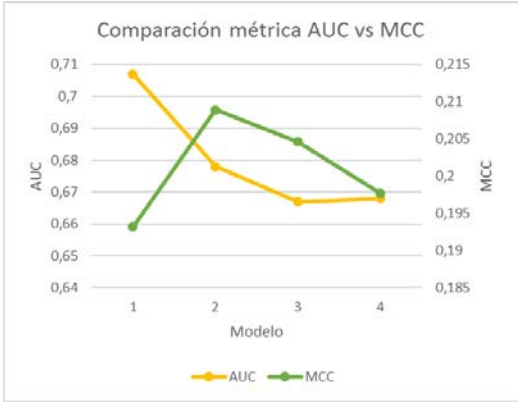


Ilustración 3 Comparación entre las métricas AUC y MCC

5 Conclusión

En este trabajo han sido expuestos 2 propuestas de métodos diferentes con el objetivo de resolver el problema de los fallos de fabricación en una línea de producción.

Se ha comprobado que la aproximación *"naive"* es la errónea y donde han surgido los problemas. Se expone en que, y por qué ha fallado, lo que creo que puede resultar muy útil para proyectos similares futuros, dado que ahora se sabe parte de lo que no hay que hacer.

La segunda aproximación se ha tratado el problema de forma más "inteligente" y se ha demostrado que técnicas han funcionado, como de bien y el porqué.

Si se hubiera dispuesto de más tiempo, se podría intentar mejorar el modelo de las siguientes maneras:

1) Aplicar un modelo de Red Neuronal, en concreto una de recurrente [27], dado la buena fama que tienen con las series de datos. En concreto las Redes Neuronales Recurrentes o RNN (las mismas siglas en inglés) son capaces de generar nuevos datos futuros a partir de la serie de datos de partida.
2) Aunque en un fondo, XGBoost es un tipo de *Ensemble method*. Sería adecuado probar de hacer un Ensemble con varios modelos a la vez. Ej: XGBoost + RNN
3) Generar nuevas características. A veces los arboles de decisión no son todo lo buenos que podrían ser con el tema de las series de datos. Por lo que nuevas características podrían ser añadidas para ayudar en este ámbito. Una manera sería haciendo métricas como la media o la desviación estándar de una característica. Por ejemplo, si una característica mantiene un valor más o menos igual, pero hay algunos que no, el árbol podría usar tanto la media como la desviación estándar como unidades para decidir.
4) Optimización de los parámetros de los modelos [28]. Un punto muy importante a la hora de generar un modelo es la correcta elección de los parámetros de estos. Por lo que sería adecuado aplicar alguna técnica para su optimización. Una técnica podría ser la *Grid Search* [29].

Las de arriba son solo algunas de las mejoras que creo que pueden aportar mejoras en la resolución del problema.

Pero por falta de tiempo solo he podido plantearlas. Otro factor es que he realizado el trabajo de forma individual, por lo que he llegado hasta donde he llegado. Ha habido gente con modelos mejores, pero iban en equipos de 4 y 5 personas, todos ellos expertos en la materia. Yo solo, con el tiempo y recursos disponibles es hasta aquí donde he podido llegar. Aunque con más tiempo sin duda aplicaría las mejoras expuestas con anterioridad.

6 Agradecimientos

Agradezco de veras el soporte que he tenido de parte de mi tutor: Dr. Ramon Baldrich Caselles. Ha sido extremadamente paciente conmigo y me ha guiado hacia la buena dirección. Dado que al principio del trabajo yo quería abarcar más de lo que el tiempo me iba a dejar, con las indicaciones cargadas de razón de Ramón ha sido posible que termine con un resultado adecuado (según mi parecer) este trabajo.

Por supuesto también me gustaría agradecer a mis padres y hermana por el soporte recibido no solo durante el transcurso de este trabajo, sino también durante toda la carrera.

7 BIBLIOGRAFÍA

[1] «Bosch Production Line Performance DATASETS,» [En línea]. Available: https://www.kaggle.com/c/bosch-production-line-performance/data.

[2] The Economist, «The Economist,» 20 10 2009. [En línea]. Available: http://www.economist.com/node/14299820. [Último acceso: 1 10 2016].

[3] E. O.-O. a. M. Roser, «OurWorldInData.org,» The Oxford University - Oxford Martin School, 2016. [En línea]. Available: https://ourworldindata.org/world-population-growth/. [Último acceso: 1 10 2016].

[4] Wikipedia contributors, «Assembly line,» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/wiki/Assembly_line. [Último acceso: 1 10 2016].

[5] Wikipedia contributors, «Quality control,» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/wiki/Quality_control. [Último acceso: 1 10 2016].

[6] International Organization for Standardization (ISO), «ISO 9000:2005, Clause 3.2.10,» [En línea]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-3:v1:en. [Último acceso: 1 10 2016].

[7] Encyclopedia Britannica, «Machine Learning,» [En línea]. Available: https://global.britannica.com/technology/machine-learning. [Último acceso: 1 10 2016].

[8] Wikipedia contributors, «Machine learning,» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/wiki/Machine_learning. [Último acceso: 1 10 2016].

[9] F. P. Ron Kohavi, «Glossary of Terms - Special Issue on Applications of Machine Learning and the Knowledge Discovery Process,» Machine Learning, 30, 271-274 (1998) - Kluwer Academic Publishers, Boston, [En línea]. Available: http://robotics.stanford.edu/~ronnyk/glossary.html.

[10] V. Castelli, «Classification And Machine Learning Glossary,» [En línea]. Available: http://www.ee.columbia.edu/~vittorio/Glossary.pdf.

[11] P. Jahnke, «Machine Learning Approaches for Failure Type Detection and Predictive Maintenance,» Technische Universität Darmstadt, 19 June 2015. [En línea]. Available: https://www.ke.tu-darmstadt.de/lehre/arbeiten/master/2015/Jahnke_Patrick.pdf.

[12] R. S. Pressman, Ingeniería del software: Un enfoque práctico, 3.ª Edición Pag. 26-30..

[13] A. Rostamizadeh y A. Talwalkar, Foundations of Machine Learning, The MIT Press ISBN 9780262018258., 2012.

[14] T. Chen, «Introduction to Boosted Trees,» University of Washington, 2014. [En línea]. Available: http://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf.

[15] F. Jerome H., «Greedy Function Approximation: A Gradient Boosting Machine,» [En línea]. Available: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf.

[16] «XGBoost GitHub Project,» [En línea]. Available: https://github.com/dmlc/xgboost. [Último acceso: 11 2016].

[17] N. V. C. K. W. B. L. O. H. y W. P. K. , «Jounal of Artificial Intelligence Research,» 2002. [En línea]. Available: http://www.jair.org/papers/paper953.html.

[18] Wikipedia contributors, «Sampling (statistics),» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/wiki/Sampling_(statistics). [Último acceso: 1 10 2016].

[19] P. A. Ihler, «CS273a: Introduction to Machine Learning,» University of California, Irvine, 2012. [En línea]. Available: http://sli.ics.uci.edu/Classes/2015W-273a?action=download&upname=09-ensembles.pdf.

[20] «Anaconda,» Continuum analytics, [En línea]. Available: https://www.continuum.io/why-anaconda.

[21] «SciPy,» [En línea]. Available: http://www.scipy.org/.

[22] «Pipes: A Brief Introduction,» The Linux Information Project, [En línea]. Available: http://www.linfo.org/pipe.html.

[23] «Matthews correlation coefficient,» wikipedia.org, [En línea]. Available: https://en.wikipedia.org/wiki/Matthews_correlation_coefficient.

[24] «The Problem of Overfitting Data,» [En línea]. Available: http://www3.cs.stonybrook.edu/~skiena/jaialai/excerpts/node16.html.

[25] W. contributors, «Curse of dimensionality,» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Curse_of_dimensionality&oldid=747289496.

[26] M. Thomas G. Tape, «The Area Under an ROC Curve,» University of Nebraska Medical Center, [En línea]. Available: http://gim.unmc.edu/dxtests/roc3.htm.

[27] A. Karpathy. [En línea]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

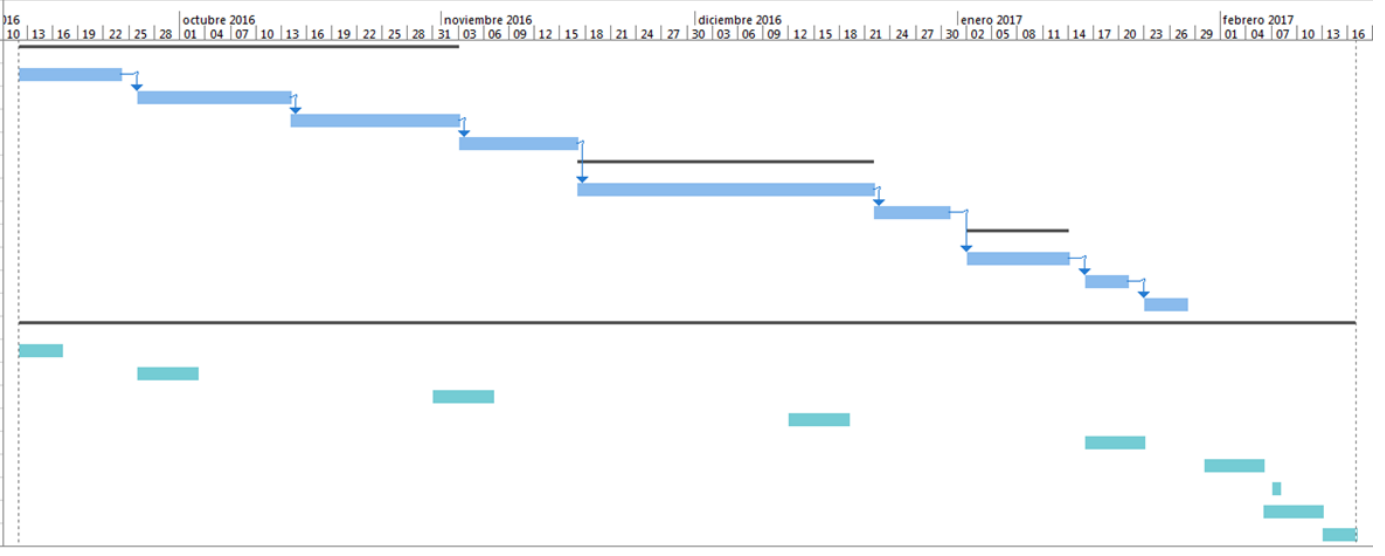
[28] W. contributors, «Hyperparameter optimization,» Wikipedia, The Free Encyclopedia., [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=760246829.

[29] J. Bergstra y Y. Bengio, « Random Search for Hyper-Parameter Optimization,» 2012. [En línea]. Available: http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf.

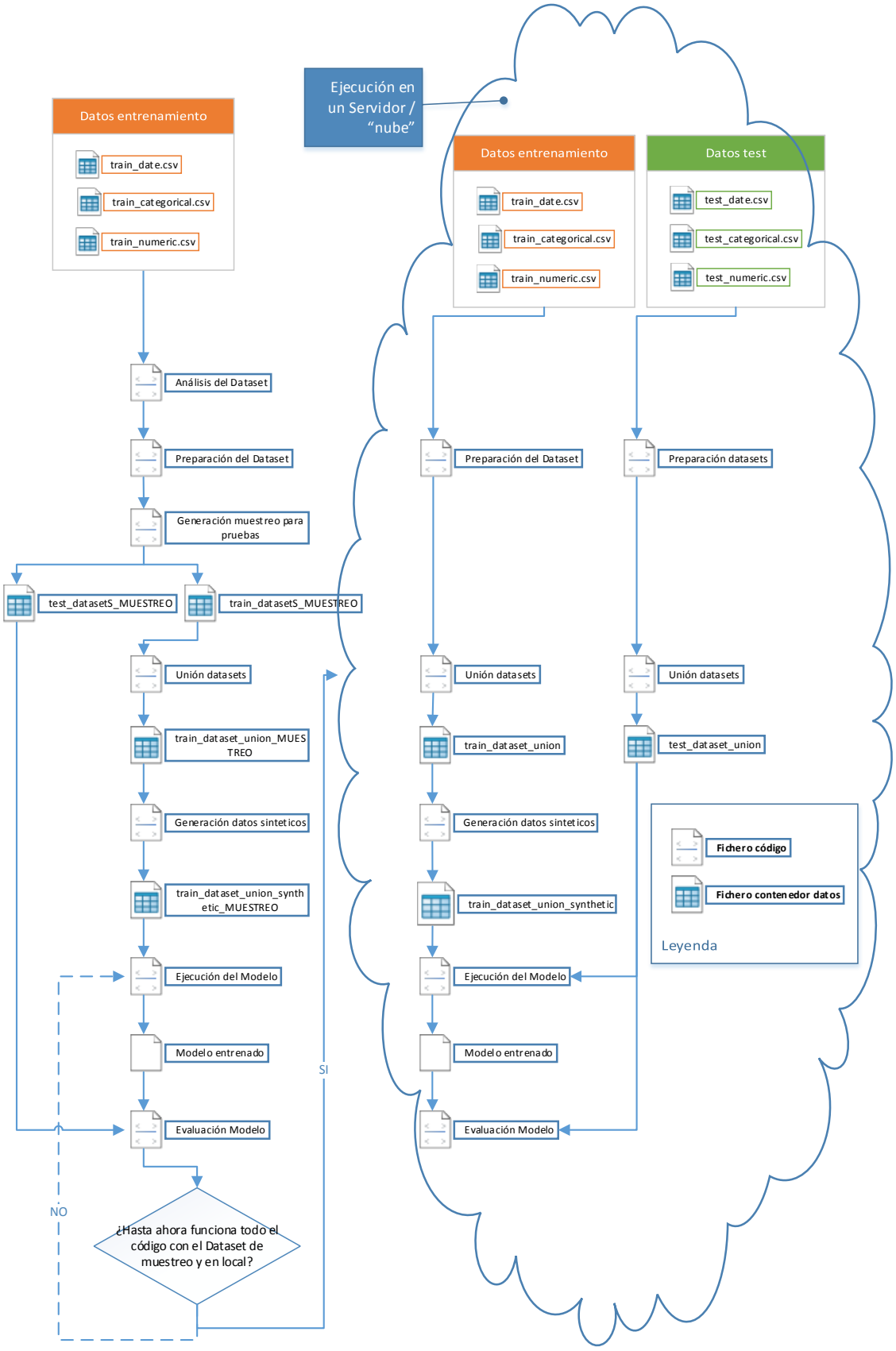
8 APÉNDICE

A1.

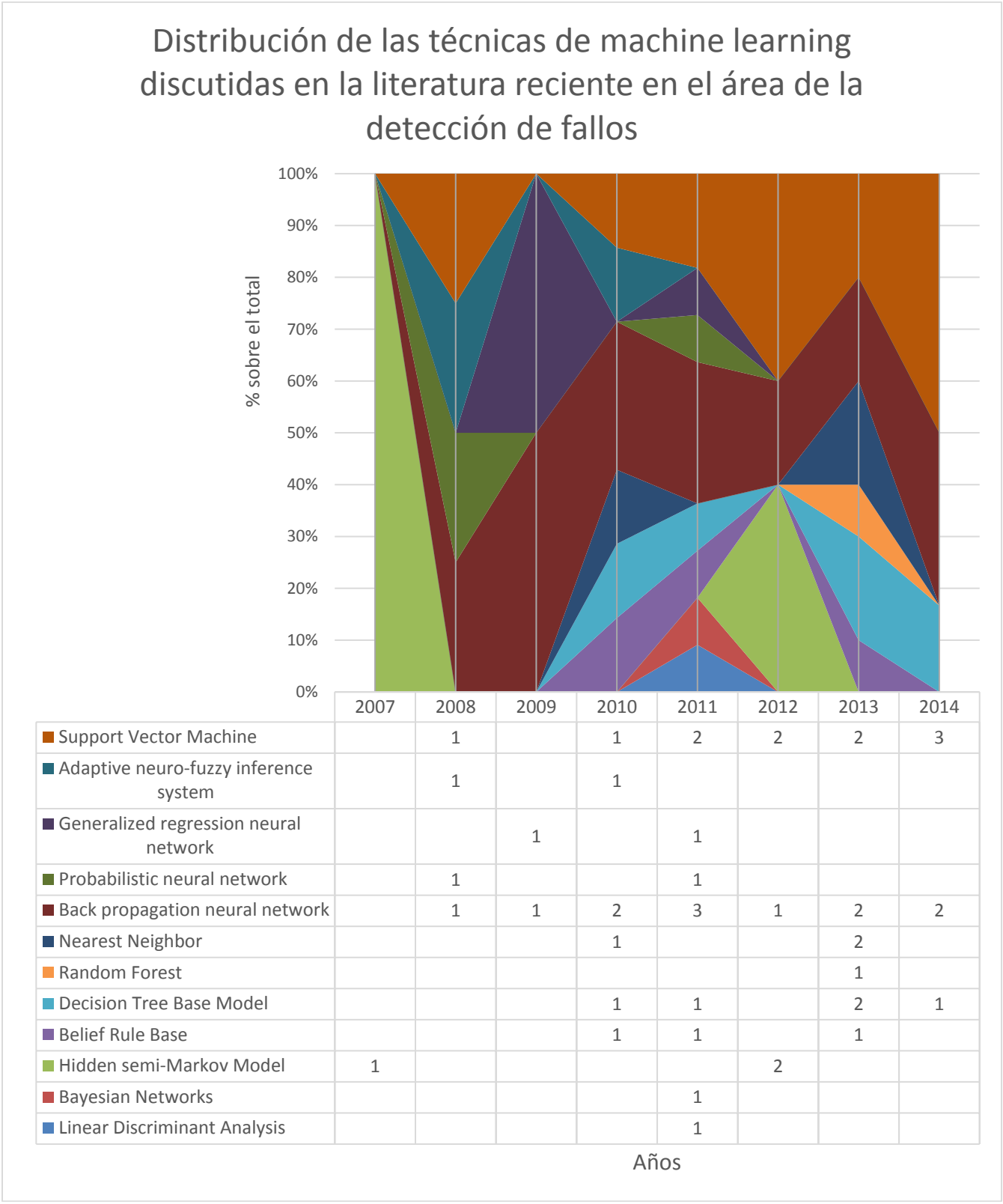
Nombre de tarea	Duración	Comienzo	Fin
Fase de Investigación	**38 días**	**lun 12/09/16**	**mié 02/11/16**
Análisis del Dataset	10 días	lun 12/09/16	vie 23/09/16
Investigación sobre cuestiones teóricas	14 días	lun 26/09/16	jue 13/10/16
Investigación sobre las tecnologías a utilizar	14 días	vie 14/10/16	mié 02/11/16
Preparación del Dataset	10 días	jue 03/11/16	mié 16/11/16
Ejecución nº1	**25 días**	**jue 17/11/16**	**mié 21/12/16**
Modelado Y Evaluación	25 días	jue 17/11/16	mié 21/12/16
Análisis siguiente modelo	7 días	jue 22/12/16	vie 30/12/16
Ejecución nº2	**10 días**	**lun 02/01/17**	**vie 13/01/17**
Modelado Y Evaluación	10 días	lun 02/01/17	vie 13/01/17
Borrador del artículo	5 días	lun 16/01/17	vie 20/01/17
Acondicionamiento del código para entregar	5 días	lun 23/01/17	vie 27/01/17
FECHAS IMPORTANTES	**114 días**	**lun 12/09/16**	**jue 16/02/17**
REUNIÓN INICIAL	5 días	lun 12/09/16	vie 16/09/16
ENTREGA - Informe Inicial	6 días	lun 26/09/16	dom 02/10/16
ENTREGA - Informe 1	6 días	lun 31/10/16	dom 06/11/16
ENTREGA - Informe 2	6 días	lun 12/12/16	dom 18/12/16
ENTREGA - Propuesta Informe Final	6 días	lun 16/01/17	dom 22/01/17
ENTREGA - Propuesta Presentación	6 días	lun 30/01/17	dom 05/02/17
ENTREGA FINAL	1 día	mar 07/02/17	mar 07/02/17
ENTREGA POSTER	6 días	lun 06/02/17	dom 12/02/17
DEFENSA TFG	4 días	lun 13/02/17	jue 16/02/17



A2.


Ejecución en un Servidor / "nube"
Datos entrenamiento
train_date.csv
train_categorical.csv
train_numeric.csv
Datos entrenamiento
train_date.csv
train_categorical.csv
train_numeric.csv
Datos test
test_date.csv
test_categorical.csv
test_numeric.csv
Análisis del Dataset
Preparación del Dataset
Generación muestreo para pruebas
Preparación del Dataset
Preparación datasets
test_datasetS_MUESTREO
train_datasetS_MUESTREO
Unión datasets
Unión datasets
Unión datasets
train_dataset_union_MUESTREO
train_dataset_union
test_dataset_union
Generación datos sinteticos
Generación datos sinteticos
train_dataset_union_synthetic_MUESTREO
train_dataset_union_synthetic
Fichero código
Fichero contenedor datos
Leyenda
Ejecución del Modelo
Ejecución del Modelo
Modelo entrenado
Modelo entrenado
SI
Evaluación Modelo
Evaluación Modelo
NO
¿Hasta ahora funciona todo el código con el Dataset de muestreo y en local?

A3



	2007	2008	2009	2010	2011	2012	2013	2014
Support Vector Machine		1		1	2	2	2	3
Adaptive neuro-fuzzy inference system		1		1				
Generalized regression neural network			1		1			
Probabilistic neural network		1			1			
Back propagation neural network		1	1	2	3	1	2	2
Nearest Neighbor				1			2	
Random Forest							1	
Decision Tree Base Model				1	1		2	1
Belief Rule Base				1	1		1	
Hidden semi-Markov Model	1					2		
Bayesian Networks					1			
Linear Discriminant Analysis					1			

Fuente del recuento de algoritmos en publicaciones: [11]