Predicción de errores en entornos de computación distribuida

Alberto Manuel Esmorís Pena
Grado de Ingeniería Informática
Inteligencia artificial

David Isern Alarcón
Carles Ventura Royo

06-2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Predicción de errores en entornos de computación distribuida*
Nombre del autor:	*Alberto Manuel Esmorís Pena*
Nombre del consultor/a:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	05/2018
Titulación::	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Español*
Palabras clave	*Inteligencia artificial, predicción de errores, centros de procesamiento de datos*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Se han utlizado técnicas actuales derivadas de la estadística (inteligencia artificial, minería de datos y aprendizaje computacional) para construir un software que clasifique los trabajos de un CPD según como se espera que sea su terminación (estado final de la ejecución).

Dicho software toma información a través de SLURM (software para la gestión de trabajos utilizado en el CPD) y la transforma a un formato adecuado (CSV) para la aplicación del algoritmo C5.0, de manera que se extraigan reglas de clasificación a partir del conjunto de datos.

Dichas reglas de clasificación son utilizadas por el software para realizar las predicciones/clasificaciones de los trabajos, que además también soporta una serie de consultas sobre los datos (para poder filtrarlos según el interés concreto) y tiene un pequeño sistema de notificaciones configurable donde se pueden consultar las predicciones de interés.

Abstract (in English, 250 words or less):

Applied modern techniques (related to Artificial Intelligence, Machine Learning and Data Mining) based in statistics to build a software capable of classifying jobs running in a Data Center attending to its termination/exit status.

The software takes data from a SLURM (Simple Linux Utility for Resource Management) environment and then transforms it into an adequate format (CSV) to use in combination with C5.0 algorithm, in order to extract classification rules from the data set.

Obtained classification rules are used by the software to classify/predict jobs. Also, it is capable of performing certain queries that work as filters that can be applied over data. Furthermore, it has a notification system which can be configured using the aforementioned queries to consider predictions of interest.

Índice

1. Introducción

1.1. Contexto y justificación del Trabajo

En los Centros de Procesamiento de Datos (CPD) donde se alquilan las infraestructuras a personal ajeno al centro, se utilizan unos sistemas de gestión de colas y recursos para asignar dichos trabajos y registrar los datos relativos a la petición, encolamiento y ejecución de estos.

Este tipo de trabajos suele corresponder a cálculos complejos que son necesarios para distinto tipo de aplicaciones: Predicción meteorológica, proyectos industriales, investigación, simulaciones físicas, docencia, etc.

Pese al sesgo común de estos trabajos (orientados a cálculos en entornos de computación distribuida), cada usuario y cada petición tienen sus propias características.

El objetivo principal de este TFG es el de construir un software que permita clasificar los trabajos atendiendo a los atributos que los definen, para conocer cual es el estado de terminación esperado de estos y con que probabilidad se predice.

De esta manera, la consecución del objetivo principal, ofrece un sistema de clasificación que permite conocer el riesgo de la ejecución de distintos trabajos. Esto puede ser utilizado de varias formas, aunque las aplicaciones derivadas escapan de los objetivos del TFG, algunos posibles serían:

- Sistema de asistencia a la toma de decisiones para elegir que trabajos aceptar o que prioridad asignar a cada trabajo en base al riesgo que entraña.

- Evaluación del riesgo de un trabajo para ajustar el coste de este (por ejemplo, trabajos que tengan cierta tendencia a fallar, podrían suponer un coste más elevado).

- Información complementaria, que se pueda integrar en otros modelos o inteligencias más complejos, para dar lugar a nuevo conocimiento. Por ejemplo, dando lugar a un sistema complejo de prevención/predicción de errores.

1.2. Objetivos del Trabajo

Considerando que el objetivo principal del TFG es la clasificación de los trabajos en base a la predicción más probable de su estado de terminación, se puede desglosar el objetivo en 4 puntos que corresponden al desarrollo cronológico del proyecto:

1. **Extracción y preparación de datos**

   El centro de procesamiento de datos trabaja con *SLURM* [1] (software para la gestión de trabajos). Por tanto, los datos se extraen de dicho entorno, utilizando las herramientas que el propio sistema ofrece para realizar consultas.

   Los atributos seleccionados para definir los trabajos son los que siguen:

   - **id_job* : ID numérico que identifica el trabajo.
   - *priority* : Entero que especifica el nivel de prioridad que se da al trabajo en el sistema de colas.
   - *account* : Cuenta desde la que se ejecuta el trabajo.
   - *user* : Usuario que solicita la ejecución del trabajo.
   - *nodes_alloc* : Número de nodos en que se ejecuta el trabajo.
   - **nodelist* : Lista de nodos en los que se ejecuta el trabajo.
   - *cpus_req* : Cantidad de CPUs solicitadas para ejecutar el trabajo.
   - *cpus_alloc* : Cantidad de CPUs implicadas en la ejecución del trabajo.
   - *mem_req* : Cantidad de memoria solicitada para ejecutar el trabajo.
   - *id_qos* : Entero que indica la clasificación, dentro del sistema de calidad del servicio, que se ha dado al trabajo.
   - *partition* : Identificador textual del conjunto de nodos que utiliza el trabajo.
   - **time_start* : Marca de tiempo que corresponde al momento en que comenzó el trabajo.
   - **time_end* : Marca de tiempo que corresponde al momento en que finalizó el trabajo (no estará disponible para todos los registros de datos, ya que los trabajos cuyo estado terminal se desea predecir no habrán concluido y por tanto, como es obvio, no tendrán un valor para este campo).
   - *timelimit* : Tiempo requerido (en segundos) para la ejecución del trabajo.
   - **time_submit* : Marca de tiempo del momento en que se presentó el trabajo.
   - **time_eligible* : Marca de tiempo del momento a partir del cual el trabajo es susceptible de ser ejecutado.
   - *time_suspended* : Cantidad de segundos que ha estado el trabajo suspendido.

- *time_elapsed* : Cantidad de segundos transcurridos desde que comenzó la ejecución del trabajo hasta su terminación (cuando se trate de procesos que ya hayan terminado, como ocurre con los datos de entrenamiento) o bien hasta el momento de recolección de datos (como ocurre en los casos de procesos que interesa predecir antes de que su ejecución haya concluido).
- *start_week_day* : Día de la semana, en formato textual, en que comenzó el trabajo.
- *start_hour* : Hora del día, representada como un número entero, en que comenzó el trabajo.
- *state* : Estado en que terminó el trabajo, se consideran los estados de SLURM: *COMPLETED*, *FAILED* y *TIMEOUT*. El resto de estados se considerarán dentro de la categoría *UNKNOWN*.

*Los campos marcados con un asterisco se consideran por ser información de interés, pero no estarán implicados en el modelo de clasificación.

2. **Análisis de datos y construcción de modelos**

Partiendo de los datos preparados, hay que realizar un análisis de los mismos que permita extraer un conjunto de reglas de clasificación que den lugar al modelo.

Para este fin, se utilizará el lenguaje de programación orientado a la estadística *R* [2] para la aplicación del algoritmo *C5.0* [3], de manera que se puedan extraer las reglas mencionadas.

Para la evaluación de los modelos construidos se utilizarán métodos estadísticos básicos, que se generan también con R. De especial interés son la *matriz de confusión*, donde se puede observar la calidad de un modelo, considerando más deseado aquel modelo que acierte más que falle en la clasificación de cada estado; la precisión del modelo, es decir, el porcentaje de casos que acierta, y el estadístico *Kappa* [4], cuyo uso puede resultar en parte polémico pues existe un debate sobre su uso por parte de los expertos en la materia, pero cuyo valor se utilizará de manera orientativa considerando que si un valor de *Kappa* es excepcionalmente bajo (no llegase al 0.1), se entendería que el modelo clasifica de una manera excesivamente arbitraria y que los resultados son fruto de la coincidencia más que de la efectividad.

3. Clasificación

Con los modelos generados deben realizarse clasificaciones sobre distintos conjuntos de datos más allá de los utilizados en su construcción, para confirmar que su funcionamiento sea el deseado.

En este punto se realizará manualmente el uso, que después se automatizará en el software de inteligencia artificial, para comprobar que el funcionamiento es correcto y que se puede satisfacer el objetivo del TFG con la estrategia de clasificación que se esté probando.

Una vez que se haya encontrado una estrategia de clasificación que funcione, podrá considerarse que este punto ha sido completado y restaría integrar el sistema de clasificación en el software de inteligencia artificial, que se desarrolla en el punto siguiente.

4. Software de inteligencia artificial

Una vez se tenga un sistema de clasificación que funcione correctamente (punto 3), basado en los modelos generados en el punto 2, que han sido construidos con los datos extraídos y preparados en el punto 1; llega el momento de integrar todas estas funcionalidades en un software escrito principalmente en *Java* [8] que realice todas estas operaciones de manera autónoma y ofrezca algunas utilidades de interés para el uso y manipulación de los datos.

Las características del software final son las que siguen:

- Recolección periódica de datos con los que construir modelos basados en reglas de clasificación, es decir, automatización del punto 1.
- Generar modelos de reglas de clasificación utilizando los datos recolectados, es decir, automatización del punto 2. Estos modelos tendrán que ser renovados periódicamente para mantener actualizado el sistema de clasificación y evitar que quede obsoleto.
- Aplicación del sistema de clasificación decidido en el punto 3, con capacidad para exportar al sistema de archivos las clasificaciones realizadas manteniendo un orden cronológico.
- Sistema de purga de datos obsoletos, mediante el cual aquellos datos que superen una antigüedad configurable sean eliminados del sistema de archivos automáticamente.
- Capacidad para realizar consultas sencillas sobre los datos:
  - Consultar trabajos cuyo estado coincida con los indicados.
  - Consultar trabajos cuyo tiempo de ejecución esté en el rango indicado.
  - Consultar trabajos asociados a los usuarios indicados.
  - Consultar trabajos asociados a las cuentas indicadas.
  - Consultar trabajos ejecutados en alguna de las particiones indicadas.

    - Consultar trabajos ejecutados por un número de nodos comprendido en el rango indicado.
    - Consultar trabajos que hayan empezado en un intervalo temporal concreto.
    - Consultar trabajos que hayan terminado en un intervalo temporal concreto.
    - Consultar trabajos que hayan comenzado en un día concreto de la semana.
    - Consultar trabajos que hayan sido ejecutados por un número de CPUs comprendido en el rango indicado.
    - Consultar trabajos que hayan solicitado una cantidad de CPUs comprendida en el rango indicado.
    - Consultar trabajos que hayan solicitado una cantidad de memoria comprendida en el rango indicado.
    - Consultar trabajos que se hayan ejecutado como mínimo en uno de los nodos indicados.
- Sistema sencillo de notificaciones que permita definir, utilizando las consultas descritas en el punto anterior, eventos de interés que serán incluidos en el directorio de notificaciones.

1.3. Enfoque y método seguido

El método de trabajo a seguir puede resumirse en dos grandes pasos: *Construcción del sistema de clasificación a mano* e *integración en un software que automatiza las tareas*.

En la primera parte, en la que se realiza el proceso a mano, el desarrollo se vale de herramientas que forman parte del propio software de gestión SLURM, como es el caso de *sacct* [5]; una utilidad para terminal que permite consultar los datos de SLURM. También se utiliza *sqoop* [6] para construir consultas SQL sobre la fuente de datos y, finalmente, se modifican los scripts en *python* [7] ofrecidos por el CPD para automatizar la extracción y preparación de datos. Dichos scripts en *python* serán utilizados también por el software final en la recolección de datos automatizada.

Una vez extraídos los datos se utiliza la herramienta *R* para construir y evaluar un modelo con estos. Cuando se encuentre una configuración de datos que permita construir un modelo cuyas clasificaciones satisfagan los criterios especificados en el objetivo segundo (punto 2 del apartado *Objetivos del Trabajo*), se realizarán experimentos clasificando con dicho modelo. Si los experimentos también ofrecen resultados adecuados, se da por concluida la primera parte y procede continuar con la integración en un software que automatice las tareas que hasta ahora se han realizado a mano o, como mucho, valiéndose de scripts.

El software final se construye utilizando los scripts en *python,* para asistir en la recolección de datos; el script en *R,* utilizado para extraer un conjunto de reglas a partir de los datos recolectados, y el resto de funcionalidades, que se implementan en *Java* apoyándose en los scripts oportunos.

El algoritmo que se ha elegido para utilizar en el script en *R* es *C5.0*, el cual es una mejora de *C4.5* que goza de bastante popularidad y resulta de fácil instalación, pues no sólo viene incluido en los repositorios de *R* sino que también existe una herramienta para el uso desde línea de comandos y se encuentra disponible en distintos repositorios. Una de las grandes ventajas de usar un algoritmo tan popular es la sencillez de instalación. En este caso concreto, para instalar el paquete con la algoritmia de *C5.0* en el CPD se ha utilizado *Anaconda Cloud* [11] (un servicio orientado al mundo de la ciencia que ofrece una amplia variedad de paquetes para *python* y *R*).

La algoritmia de *C5.0* se basa en el algoritmo *ID3* [12] para la extracción de reglas. Existen otros algoritmos similares que podrían utilzarse en lugar del popular C5.0, por ejemplo: *Algoritmo de cobertura* y *ADTree* (árbol de decisión alternativo); ambos basados en *ID3* también. Además, existen otras alternativas como *JRip* [13], *Nnge* [14] , *OneR* [15] y *Ridor* [16].

1.4. Planificación del Trabajo

ITERACIONES

ITERACIÓN	OBJETIVOS
1	1.1 Programación de scripts para la extracción y transformación de datos 1.2 Construcción de modelos con los datos extraídos. 1.3 Elección del modelo/s más adecuados
2	2.1 Probar el modelo con los datos de prueba 2.2 Probar el modelo con nuevos datos
3	3.1 Desarrollo base del software de inteligencia artificial en *Java*. 3.2 Integración del modelo predictivo en la IA.
4	4.1 Añadir funciones de consulta al software. 4.2 Añadir sistema de notificaciones al software.
5	5.1 Puesta a punto de la IA y de sus distintas funcionalidades, solventando los errores que puedan surgir en el proceso.

CALENDARIO

ITERACIÓN	PLAZO ESTIMADO
1.1	De 12-03-2018 a 25-03-2018
1.2	De 19-03-2018 a 01-04-2018
1.3	De 26-03-2018 a 01-04-2018
2.1	De 02-04-2018 a 08-04-2018
2.2	De 02-04-2018 a 08-04-2018
3.1	De 09-04-2018 a 29-04-2018
3.2	De 23-04-2018 a 06-05-2018
4.1	De 30-04-2018 a 13-05-2018
4.2	De 07-05-2018 a 13-05-2018
5.1	De 07-05-2018 a 20-05-2018

DETALLE

1.1. Programación de scripts para la extracción y transformación de datos

En primer lugar se realiza un estudio preliminar de los datos utilizando las herramientas ya mencionadas (*sacct* y *sqoop*). Una vez que se han entendido los datos se modifica un script en *python* ofrecido por el CPD para que trabaje con el conjunto de datos elegido.

Al término de este hito se deberá disponer de un script cuya invocación permita extraer los datos ya transformados en un archivo en formato csv (valores separados por comas). No obstante, posteriores transformaciones pueden ser realizadas sobre los datos para ajustarlos a las demandas de la algoritmia que se vaya aplicar. En cualquier caso, estas transformaciones serían menores y consistirían en suprimir registros o ignorar atributos, pero siempre respetarían el formato csv generado por el script de extracción.

1.2. Construcción de modelos con los datos extraídos.

Utilizando la herramienta R se aplicará la algoritmia C5.0 para extraer un conjunto de reglas que constituyan un modelo de clasificación capaz de operar sobre los datos que definen los trabajos del CPD.

Se probarán distintos modelos atendiendo al volumen de datos recolectado. Es decir, se probará a entrenar un modelo con datos de un día, de una semana, de un mes, o de distintos rangos temporales. Se tratará de conseguir un modelo o conjunto de modelos que permitan tener un sistema eficiente y actualizado. Esto es de especial interés ya que utilizar un conjunto de datos que corresponda a un intervalo de tiempo demasiado grande podría llevar a que el modelo se base en una cantidad excesiva de datos erróneos/obsoletos, que correspondan a configuraciones pasadas del CPD altamente divergentes respecto de la realidad presente (cambios de hardware, de usuarios, de software de gestión utilizado, etc.)

1.3. Elección del modelo/s más adecuados

Con el fin de elegir el modelo (o modelos, si se optase por utilizar varios) más adecuado, se valorarán los siguientes criterios:

1. *Volumen de datos necesario*. Se considera óptimo obtener el modelo más preciso con el menor volumen de datos necesario para ello. Por ejemplo, si con los datos de 2 semanas se puede construir un modelo con una precisión alta, se preferirá a uno de 3 semanas.
2. *Antigüedad no excesiva*. Como se ha comentado en el detalle de la iteración 1.2, utilizar datos muy antiguos puede dar lugar a un modelo que realice apreciaciones que no se correspondan con la realidad presente, ya que los CPD están sujetos a cambios de diversa índole a lo largo de los años (distinta configuración, distinto software de gestión, cambios de hardware, distintos usuarios, etc.)

3. *Tiempo de construcción del modelo*. Es importante que el modelo se pueda construir en un intervalo de tiempo razonable. Si el volumen de datos a utilizar implica que la extracción del conjunto de reglas tome un tiempo excesivo, dicho modelo será considerado menos deseable que otro que tome menos tiempo y ofrezca una eficiencia semejante.

2.1. Probar el modelo con los datos de prueba

Se utilizará un conjunto de datos de prueba para evaluar el rendimiento del modelo generado. El criterio a tener en cuenta para la evaluación del modelo es el que sigue:

1. *Matriz de confusión*. Se considerará más apto un modelo cuando su matriz de confusión tenga más aciertos que fracasos en la clasificación de los distintos estados posibles, con la posible excepción del estado *UNKNOWN*; ya que estos casos constituyen un conglomerado de menor interés.
2. *Precisión*. A mayor precisión del modelo (donde por precisión se entiende tasa de acierto, que se obtiene dividiendo el número de aciertos entre el total de clasificaciones realizadas), más deseable se considerará este.
3. *Estadístico Kappa*. Como se ha comentado anteriormente, el uso de este estadístico está sujeto a cierta discusión. En este TFG se utilizará de manera orientativa considerando que un valor de Kappa excesivamente bajo (por debajo del 0.1) implica que el modelo clasifica con demasiada arbitrariedad.
4. *Amplia cobertura de usuarios y otros datos categóricos*. En los datos utilizados conviven atributos categóricos y numéricos. Algunos atributos categóricos abarcan una gran cantidad de valores posibles, como el que hace alusión a los usuarios. Aquellos valores que no figuren en el conjunto de datos de entrenamiento no serán contemplados por el modelo y posteriormente no podrán ser clasificados. Por tanto, un modelo que cubra gran variedad de valores posibles para estos atributos de amplio espectro, será más deseable que uno que cubra menos.

2.2. Probar el modelo con nuevos datos

Considerando los mismos criterios que con el conjunto de datos de prueba, se realizarán experimentos sobre distintos conjuntos de datos para comprobar que el modelo clasifica correctamente.

3.1. Desarrollo base del software de inteligencia artificial en Java.

En esta iteración se desarrolla la base fundamental del software para poder interactuar con los distintos componentes que luego habrá que integrar:

1. *Capacidad para trabajar con archivos CSV*
   Debe poderse importar y exportar un archivo CSV, así como manipularlo y consultarlo dentro del propio software.

   El código necesario para estas funcionalidades estará organizado en el paquete: *ai.lazuli.utils.csv*

2. *Control de archivos*
   El software debe poder purgar los archivos que superen cierta antigüedad.

   El código necesario para estas funcionalidades estará organizado en los paquetes: *ai.lazuli.utils.files* y *ai.lazuli.cesga.data*

3. *Sistema de logs*
   Con el fin de dejar un registro de los eventos de interés que suceden durante la ejecución del software, se implementará un sencillo sistema de logs.

   El código necesario para estas funcionalidades estará organizado en el paquete: *ai.lazuli.cesga.logging*

4. *Recolección de datos*
   Para poder utilizar los scripts de recolección de datos será necesario implementar un *wrapper* que permita controlar dichas operaciones desde el software.

   El código necesario para estas funcionalidades estará organizado en el paquete: *ai.lazuli.cesga.data.collection*

5. *Configuración*
   Ciertos parámetros que definen los comportamientos del software estarán externalizados en un archivo de configuración que la aplicación debe ser capaz de gestionar.

   El código necesario para estas funcionalidades estará contenido en una única clase: *ai.lazuli.cesga.Config*

6. *Acceso sincronizado*
   Con el fin de garantizar la correcta gestión del acceso a los recursos compartidos, en este caso los archivos que contienen los modelos y que pueden ser escritos y leídos por distintos componentes del software, se creará una clase para asistir en el acceso concurrente a estos recursos.

   El código necesario para esta funcionalidad estará contenido en una única clase: *ai.lazuli.cesga.data.ModelSynchronizer*

3.2. Integración del modelo predictivo en la IA.

El modelo predictivo que se construye con *R* debe ser utilizable por el software en *Java*. Para este fin, el script en *R* exportará el modelo a un archivo que será leído e interpretado por la aplicación.

El código necesario para generar el modelo utilizando el script en *R* estará contenido en el paquete: *ai.lazuli.cesga.data* y *ai.lazuli.cesga.data.collection*

El código necesario para importar y aplicar las reglas de clasificación estará contenido en el paquete: *ai.lazuli.utils.classificationrules*

4.1. Añadir funciones de consulta al software.

El software debe permitir realizar una serie de consultas (definidas en el punto 4 de *Objetivos del Trabajo*) que sirvan para filtrar los registros de datos según un interés concreto.

El código necesario para estas funcionalidades estará implementado en la clase *ai.lazuli.cesga.data.Querier* que a su vez utilizará la clase *ai.lazuli.cesga.data.QuerierException* para realizar el control de excepciones oportuno.

4.2. Añadir sistema de notificaciones al software.

El sistema de notificaciones se basa en la existencia de un notificador configurable que, utilizando las consultas anteriormente mencionadas, exporta los datos de interés.

El código necesario para esta funcionalidad estará implementado en la clase *ai.lazuli.cesga.data.Notifier* que a su vez utilizará la clase *ai.lazuli.cesga.data.NotifierException* para realizar el control de excepciones oportuno.

5.1. Puesta a punto de la IA y de sus distintas funcionalidades, solventando los errores que puedan surgir en el proceso.

Con el tiempo que reste se pulirán los errores que se hayan observado en el código y, de ser posible, se desplegará en las infraestructuras del CPD para analizar su comportamiento.

1.5. Breve sumario de productos obtenidos

El producto final de mayor interés será el **software en *Java*** que realizará las funciones de recolección de datos, extracción de un conjunto de reglas a partir de estos datos, construcción de un modelo de clasificación con dichas reglas e integración; dando lugar a un sistema capaz de evaluar los distintos trabajos que gestiona el CPD, para predecir cual es el estado de terminación más probable.

A su vez, este producto se descompone en:

- Componente en *python* para la extracción y transformación inicial de datos.
- Componente en *R* para la construcción de un modelo de clasificación basado en reglas extraídas de los datos.
- Código en *Java* que integra el resto de componentes y añade sus propias funcionalidades.

1.6. Breve descripción de los otros capítulos de la memoria

Los capítulos en que se detalla el proceso de elaboración del producto final son los que siguen:

2.1. Extracción y preparación de datos

En este capítulo se detalla el proceso de extracción y preparación de los datos. Se explica desde el proceso inicial de toma de contacto con las herramientas ya existentes hasta la culminación en la automatización del proceso a través de *scripting*.

2.2. Construcción y evaluación de modelos

En este capítulo se explica tanto el script en *R* como la metodología de trabajo seguida para escoger los modelos más adecuados.

2.3. Software final

En este capítulo se expone el software final construido y el desglose de sus funcionalidades.

2.4. Resultados

En este capítulo se presentan y analizan los resultados obtenidos, así como los scripts utilizados para tal fin.

2. Predicción de errores en entornos de computación distribuida

2.1. Extracción y preparación de datos

EXTRACCIÓN DE LOS DATOS

En un primer lugar se utiliza la herramienta *sacct* para consultar los trabajos con estado *FAILED* a partir del 1 de enero de 2017, la instrucción para tal fin es:

```
sacct -a --nnodes=1 -s FAILED -S 2017-01-01 --
format=JobID,JobIDRaw,Account,User,UID,Cluster,NNodes,NodeList,Start,End,
ExitCode,State > new_failed_jobs.txt
```

Con esto se genera un archivo (*new_failed_jobs.txt*) que contiene registros correspondientes a distintos trabajos, donde resultan de especial interés el *JobID*, para poder identificar el trabajo, y el *State*, para conocer el estado categórico del trabajo. Los otros campos sirven para tomar un primer contacto con los usuarios, clusters, número de nodos que ejecuta un trabajo, listado de nodos, etc.

A continuación, se consulta la base de datos utilizando *sqoop-eval*, para conocer la equivalencia entre el código numérico y los estados categóricos que muestra en su salida la herramienta *sacct*. Para ello se ejecuta la siguiente consulta:

```
sqoop-eval --connect 'jdbc:mysql://10.118.67.106/slurm_acct_db' --username
<usuario> --password <password> --query 'SELECT job_db_inx, id_job,
time_start, time_end, nodelist, nodes_alloc, state, exit_code FROM
ft2_job_table WHERE nodelist != "None assigned" and exit_code != 0 and
nodes_alloc > 1 and time_end > 1483228800'
```

De la salida obtenida se buscan en el anterior archivo los trabajos por su identificador, para establecer así la asociación entre el entero que se almacena en la base de datos y el estado categórico, quedando la asociación tal que:
1 corresponde al estado *RUNNING*
3 corresponde al estado *COMPLETED*
4 corresponde al estado *CANCELLED*
5 corresponde al estado *FAILED*
6 corresponde al estado *TIMEOUT*

Tras observar la distribución de los datos y reflexionar acerca del modelo que se desea construir, se decide considerar los estados 3 (*COMPLETED)*, 5 (*FAILED*) y 6 (*TIMEOUT*). El resto de estados se agrupará bajo el valor categórico *UNKNOWN*, ya que no se considera relevante distinguir entre unos y otros.

De la decisión tomada, se entiende necesario justificar el descarte del estado *CANCELLED*. Podría haber cierto interés en predecir cuando los usuarios cancelan trabajos. No obstante, teniendo en cuenta que las operaciones de cancelación pueden venir tanto por parte del usuario que lanzó el trabajo, como de un técnico del propio CPD, la situación se complica. Por otro lado, el objetivo principal parte de la idea de clasificar los trabajos según si van a fallar o no, es discutible que una cancelación manual sea un fallo. Por tanto, al no formar parte del interés principal, presentar complejidades extra (distinguir si la cancelación fue por parte del usuario o no y el hecho de que las cancelaciones pueden ser realizadas por motivos de muy diversa índole) y observarse un aumento no deseado de la entropía en algunos experimentos de construcción de modelos predictivos al considerar el estado *CANCELLED* emancipado de la categoría *UNKNOWN*; se decide prescindir de la consideración independiente de este estado en los modelos.

Partiendo de lo anteriormente expuesto, se modifica el script en *python* ofrecido por el CPD para extraer los datos de interés para el proyecto. Dicho script se adjunta como anexo 1.

Finalmente, a partir del análisis previamente realizado se extraen los siguientes datos de entrenamiento, agrupados en conjuntos de datos según el intervalo temporal al que corresponden:

training_week1.csv : De 29 de Enero 00:00 hasta 5 de Febrero 00:00
training_week2.csv : De 5 de Febrero 00:00 hasta 12 de Febrero 00:00
training_week3.csv : De 12 de Febrero 00:00 hasta 19 de Febrero 00:00
training_month1.csv : De 1 de Enero 00:00 hasta 31 de Enero 00:00
training_month2.csv : De 31 de Enero 00:00 hasta 2 de Marzo 00:00

test_day1.csv : De 15 de Febrero 00:00 hasta 16 de Febrero 00:00
test_day2.csv : De 5 de Marzo 00:00 hasta 6 de Marzo 00:00
test_day3.csv : De 10 de Marzo 00 hasta 11 de Marzo 00:00
test_week1.csv : De 5 de Febrero 00:00 hasta 12 de Febrero 00:00
test_week2.csv : De 2 de Marzo 00:00 hasta 9 de Marzo 00:00
test_month1.csv : De 8 de Febrero 00:00 hasta 10 de Marzo 00:00
test_month2.csv : De 2 de Marzo 00:00 hasta 1 de Abril 00:00
test_aggregate.csv : De 1 de Enero 00:00 hasta 1 de Abril 00:00

Con el fin de facilitar la replicabilidad del experimento, se adjunta el script en *bash* que se ha utilizado para programar los distintos lanzamientos del script en *python* que extrae los datos:

```
#!/bin/bash

TRAINING_DIR='training_data'
TEST_DIR='test_data'
PYTHON_SCRIPT='export_metrics.py'

mkdir -p $TRAINING_DIR
mkdir -p $TEST_DIR

# TRAINING DATA EXTRACTION #
# ------------------------ #
# Jan 29 to Feb 5
python $PYTHON_SCRIPT -out $TRAINING_DIR'/training_week1.csv' -started-after-ts 1517184000 -started-before-ts 1517788800
# Feb 5 to Feb 12
python $PYTHON_SCRIPT -out $TRAINING_DIR'/training_week2.csv' -started-after-ts 1517788800 -started-before-ts 1518393600
# Feb 12 to Feb 19
python $PYTHON_SCRIPT -out $TRAINING_DIR'/training_week3.csv' -started-after-ts 1518393600 -started-before-ts 1518998400
# Jan 1 to Jan 31
python $PYTHON_SCRIPT -out $TRAINING_DIR'/training_month1.csv' -started-after-ts 1514764800 -started-before-ts 1517356800
# Jan 31 to Mar 2
python $PYTHON_SCRIPT -out $TRAINING_DIR'/training_month2.csv' -started-after-ts 1517356800 -started-before-ts 1519948800


# TEST DATA EXTRACTION #
# -------------------- #
# Feb 15 to Feb 16
python $PYTHON_SCRIPT -out $TEST_DIR'/test_day1.csv' -started-after-ts 1518652800 -started-before-ts 1518739200
# Mar 5 to Mar 6
python $PYTHON_SCRIPT -out $TEST_DIR'/test_day2.csv' -started-after-ts 1520208000 -started-before-ts 1520294400
# Mar 10 to Mar 11
python $PYTHON_SCRIPT -out $TEST_DIR'/test_day3.csv' -started-after-ts 1520640000 -started-before-ts 1520726400
# Feb 5 to Feb 12
python $PYTHON_SCRIPT -out $TEST_DIR'/test_week1.csv' -started-after-ts 1517788800 -started-before-ts 1518393600
# Mar 2 to Mar 9
python $PYTHON_SCRIPT -out $TEST_DIR'/test_week2.csv' -started-after-ts 1519948800 -started-before-ts 1520553600
# Feb 8 to Mar 10
python $PYTHON_SCRIPT -out $TEST_DIR'/test_month1.csv' -started-after-ts 1518048000 -started-before-ts 1520640000
# Mar 2 to Apr 1
python $PYTHON_SCRIPT -out $TEST_DIR'/test_month2.csv' -started-after-ts 1519948800 -started-before-ts 1522540800
# Aggregate
python $PYTHON_SCRIPT -out $TEST_DIR'/test_aggregate.csv' -started-after-ts 1514764800 -started-before-ts 1522540800
```

PREPARACIÓN DE LOS DATOS

En lo tocante a la preparación de datos, los atributos considerados mediante el script de extracción están ya preparados en cuanto a su formato. Los atributos categóricos están traducidos a un conjunto limitado de valores, incluso para los campos en que existe una gran cantidad de categorías posibles, como es el caso de los usuarios. Respecto a los estados, estos son traducidos de formato numérico a las categorías oportunas (según la relación ya mencionada), por lo que se consideran también como categóricos. Por otro lado, los atributos numéricos se extraen correctamente y no necesitan ser revisados de ninguna manera.

El principal problema que se enfrenta en la preparación de datos no reside en el formato de estos sino en ciertos registros, que debido a atributos categóricos como el usuario que cubren un amplio espectro de valores posibles, deben ser tratados con especial cuidado.

Antes de entrar en la fase de construcción del modelo a partir de los conjuntos de datos de entrenamiento, se han llevado a cabo una serie de operaciones de preparación de datos; tanto para los datos de entrenamiento como para los de prueba.

En primer lugar, de los datos extraídos se ha considerado un estado que se denomina *CANCELLED*, pero que no será incluido en el modelo predictivo. Los registros que correspondan a dicho estado han de ser eliminados, ya que no resulta de interés tenerlos en cuenta.

Este estado corresponde a trabajos que han sido cancelados por el usuario, lo cual es una decisión relativamente arbitraria que introduce una cantidad de entropía indeseada en el modelo. Además, tampoco es de interés predecir cuando un usuario va a decidir cancelar un trabajo; principalmente porque los diferentes motivos de cancelación dificultan la comprensión de este fenómeno y, además, una parte de los trabajos cancelados corresponde a las actuaciones de los técnicos del centro de procesamiento de datos; operación que no se pretende predecir tampoco.

Para preparar los datos se utilizan dos scripts, en primer lugar un sencillo script en *bash*, que recibe como primer parámetro la ruta al fichero de datos de entrenamiento original y, como segundo parámetro la ruta al fichero que se quiere generar. Dicho script elimina los registros cuyo estado corresponda a *CANCELLED*:

```
#!/bin/bash

# Fixing a CSV consists in removing those rows which have state = CANCELLED
# since this state introduces undesired entropy in the data

if [ $# -lt 1 ]; then
        echo 'Specify the path to the csv file to fix (remove CANCELLED state rows)'
fi

csvToFix=$1
outputCsv="$csvtoFix".fixed
if [ $# -gt 1 ]; then
        outputCsv=$2
fi

cat $csvToFix | grep -v 'CANCELLED' > $outputCsv
```

Las invocaciones que se han realizado al script (que se encuentra en el archivo *fix.sh*) *son* las que siguen:

```
./fix.sh training_week1.csv training_week1_fixed.csv
./fix.sh training_week2.csv training_week2_fixed.csv
./fix.sh training_week3.csv training_week3_fixed.csv
./fix.sh training_month1.csv training_month1_fixed.csv
./fix.sh training_month2.csv training_month2_fixed.csv
```

A mayores, se utiliza otro script, algo más complejo y escrito en *python,* para purgar los registros de los datos de prueba que presentan valores categóricos no contenidos en los registros de los datos de entrenamiento. Esto incluirá todos aquellos registros cuyo estado sea *CANCELLED*. Dicho script es el que sigue:

```
# *************** #
# *   V A R S   * #
# *************** #
input_files = [
        '<path to test data 1>',
        '<path to test data 2>',
        '<path to test data n>'
]
input_reference = '<path to reference training data file>'

user_index = 3 # Index for user value in CSV records (0 is the first one)
account_index = 2 # Index for account value in CSV records
partition_index = 10 # Index for partition value in CSV records
start_week_day_index = 18 # Index for start_week_day value in CSV records
state_index = 20 # Index fot state value in CSV records

# *************** #
# *   M A I N   * #
# *************** #
if __name__ == '__main__':
        # Build sets from input_reference (training data)
        users = set()
        accounts = set()
        partitions = set()
        start_week_days = set()
        states = set()
        fin = open(input_reference,"r")
        print('Parsing input reference: "{}"  ...'.format(input_reference))
        fin.readline() # Skip header
        line = fin.readline()[:-1]
        while len(line) > 0:
                split = line.split(",");
                users.add(split[user_index])
                accounts.add(split[account_index])
                partitions.add(split[partition_index])
                start_week_days.add(split[start_week_day_index])
                states.add(split[state_index])
                line = fin.readline()[:-1]
        fin.close()
        try:
                states.remove('CANCELLED')
        except KeyError: pass

        # Fix test data
        for infile in input_files:
                count = 0
                fin = open(infile, "r")
                outfile = infile.replace(".csv","_fixed.csv")
                fout = open(outfile, "w")
                print('Generating output: "{}"  ...'.format(outfile))
                fout.write(fin.readline()) # Copy header
                line = fin.readline()[:-1]
                while len(line) > 0:
                        split = line.split(",")
                        if (split[user_index] in users and
                                        split[account_index] in accounts and
                                        split[partition_index] in partitions and
                                        split[start_week_day_index] in start_week_days and
                                        split[state_index] in states):
                                fout.write('{}\n'.format(line))
                        else : count += 1
                        line = fin.readline()[:-1]
                fin.close()
                fout.close()
                print('For test data in file "{file:}" {num:} records were purged'
                        .format(file=infile, num=count))
```

El uso del script es relativamente sencillo. En la primera parte del mismo existen una serie de variables, de estas variables hay dos en concreto que son las que se explicarán aquí:

> *input_files* : Lista con los archivos de datos de prueba a purgar.
> *input_reference* : Ruta al archivo de entrenamiento que se usará como referencia para conocer los valores categóricos válidos que deben contener los registros a conservar.

Para utilizar el script, basta con modificar dichas variables para que considere los archivos deseados.

Tras la aplicación de estos scripts, ambos conjuntos de datos quedan preparados para ser utilizados en R.

2.2. Construcción y evaluación de modelos

SCRIPT *R*

El script en *R* que se ha elaborado para construir y evaluar modelos es el que sigue:

```
# C5.0 for decission trees and rule-based models

# Libraries
library(C50)
library(caret)

# Data importing (without id field)
training_data = read.csv('<path to training data>', header = TRUE)[,-6][,-11:-12][,-12:-13][,-1]
test_data = read.csv('<path to test data>', header = TRUE)[,-6][,-11:-12][,-12:-13][,-1]
levels(test_data$state) = levels(training_data$state)
colnames(training_data)

# Build model
model <- C5.0(x = training_data[,-length(training_data)], y = training_data$state, rules = TRUE,
trials=1)
model
summary(model)
model$boostResults
write(model$rules, file="/<path>/model.rules")

# Predictions
predictions <- predict(model, newdata = test_data, na.action = na.pass)
expected = test_data$state
result <- expected == predictions
ok = length(result[result==TRUE])
err = length(result[result==FALSE])
acc = ok/length(result)
cat('OK predictions: ', ok, '\nFailed predictions: ', err,
    '\nPredictions accuracy: ', acc)
confusionMatrix(predictions, expected)
```

En primer lugar se cargan las librerías, utilizando la función *library*:

- **C50** : Es la librería que contiene la algoritmia necesaria para extraer las reglas de clasificación.
- **caret** [9] : Es una librería de utilidades escogida para asistir en el trabajo con modelos predictivos. Concretamente se importa para utilizar la función *confusionMatrix*.

Después, se importan los datos de entrenamiento y prueba, con la función *read.csv*, nótese que la marcas *<path to training data>* y *<path to test data>* deben ser sustituidas por las rutas correspondientes. Además, se ajustan los niveles categóricos del estado para los datos de prueba, de manera que coincidan con los de entrenamiento.

La instrucción *colnames(training_data)* está ahí para conocer los nombres de las columnas finales, puesto que al importar los datos se han truncado columnas que no eran de interés para la construcción del modelo; principalmente aquellas correspondientes a marcas de tiempo como *time_end*, *time_start*, *time_submit*, *time_eligible* y, también, el id del trabajo.

Tras esto, se construye el modelo con la función *C5.0*. El parámetro *x* corresponde a los datos de entrenamiento; excluyendo el atributo *state* que será enviado como parámetro *y,* ya que es el estado lo que interesa predecir. El parámetro *trials* corresponde al número de iteraciones del algoritmo C5.0. En caso de realizar varias iteraciones, las que vienen después de la primera se centrarían en aquellos subconjuntos de datos cuya clasificación es menos fiable o precisa y les daría un trato especializado [10].

Las siguientes instrucciones son simples impresiones para conocer el modelo, hasta que se llega a la función *write*, donde se exporta el modelo a un archivo que contiene las reglas y que posteriormente será el que importe el software final escrito en *Java*.

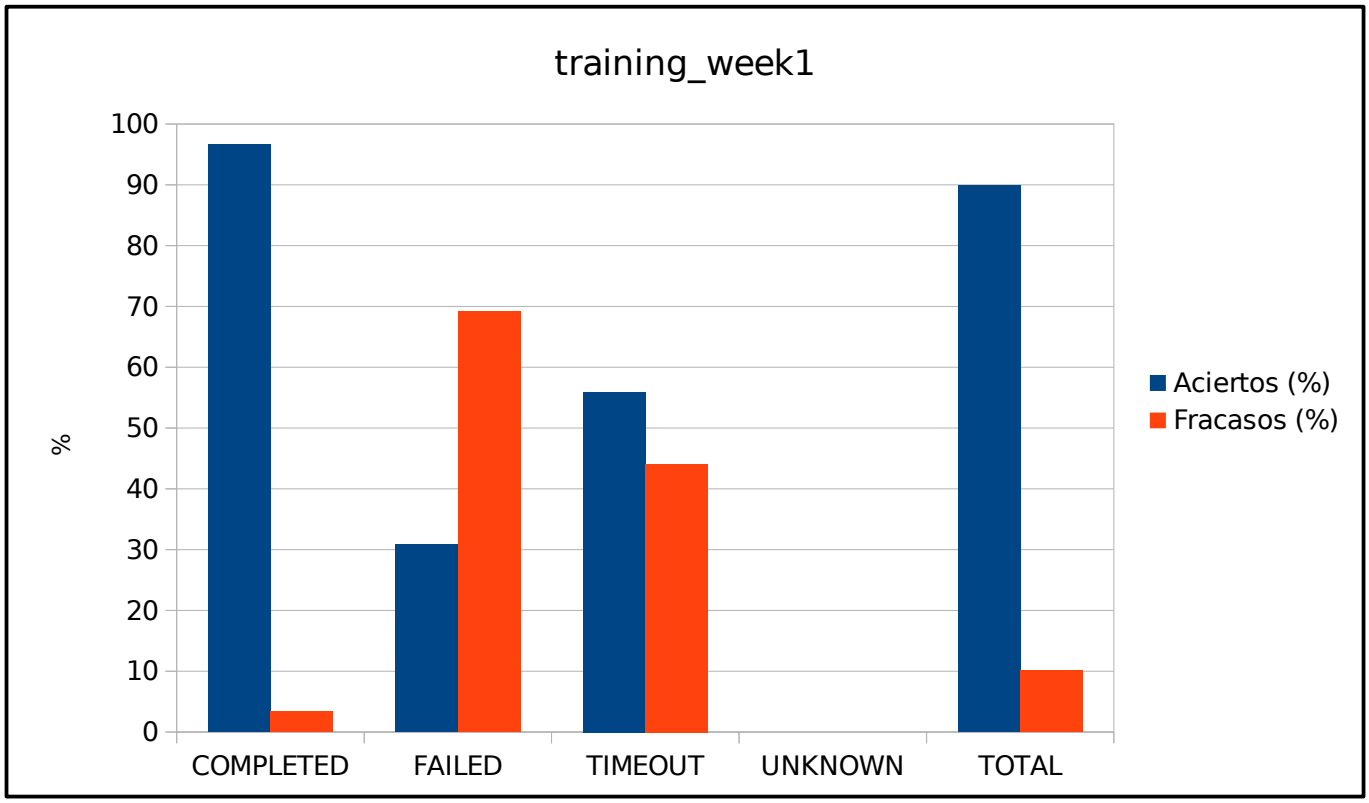
A continuación, se utiliza la función *predict* para clasificar los datos de prueba con el modelo que se ha construido a partir de los de entrenamiento. Cabe destacar que el parámetro *na.action* se fija a *na.pass* para evitar problemas en la gestión de registros en los que pueda faltar algún dato.

Finalmente, se generan unas métricas sencillas, de manera manual, para conocer el número de aciertos y fracasos en las predicciones y se imprime la matriz de confusión.

EXPERIMENTOS

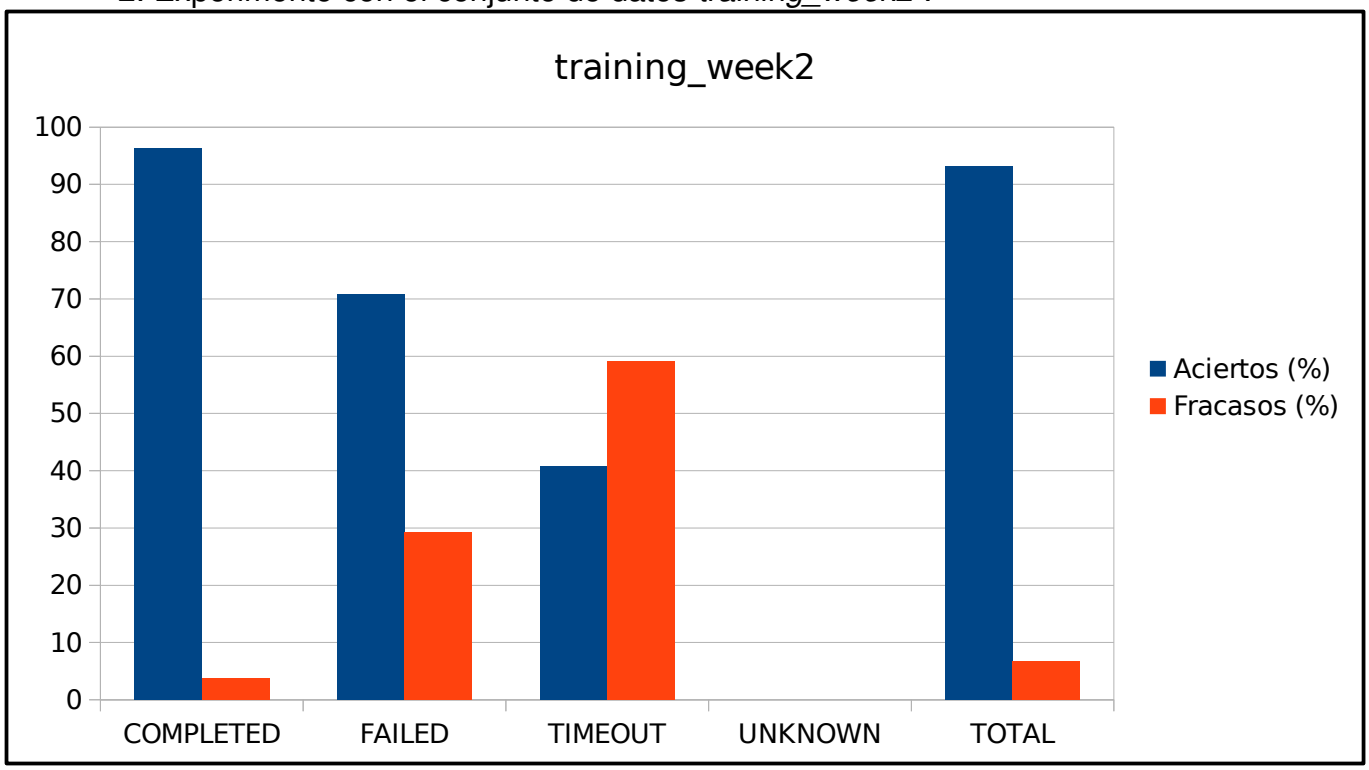
(Se adjuntan los datos de los experimentos en la sección de anexos)

1. Experimento con el conjunto de datos *training_week1* :



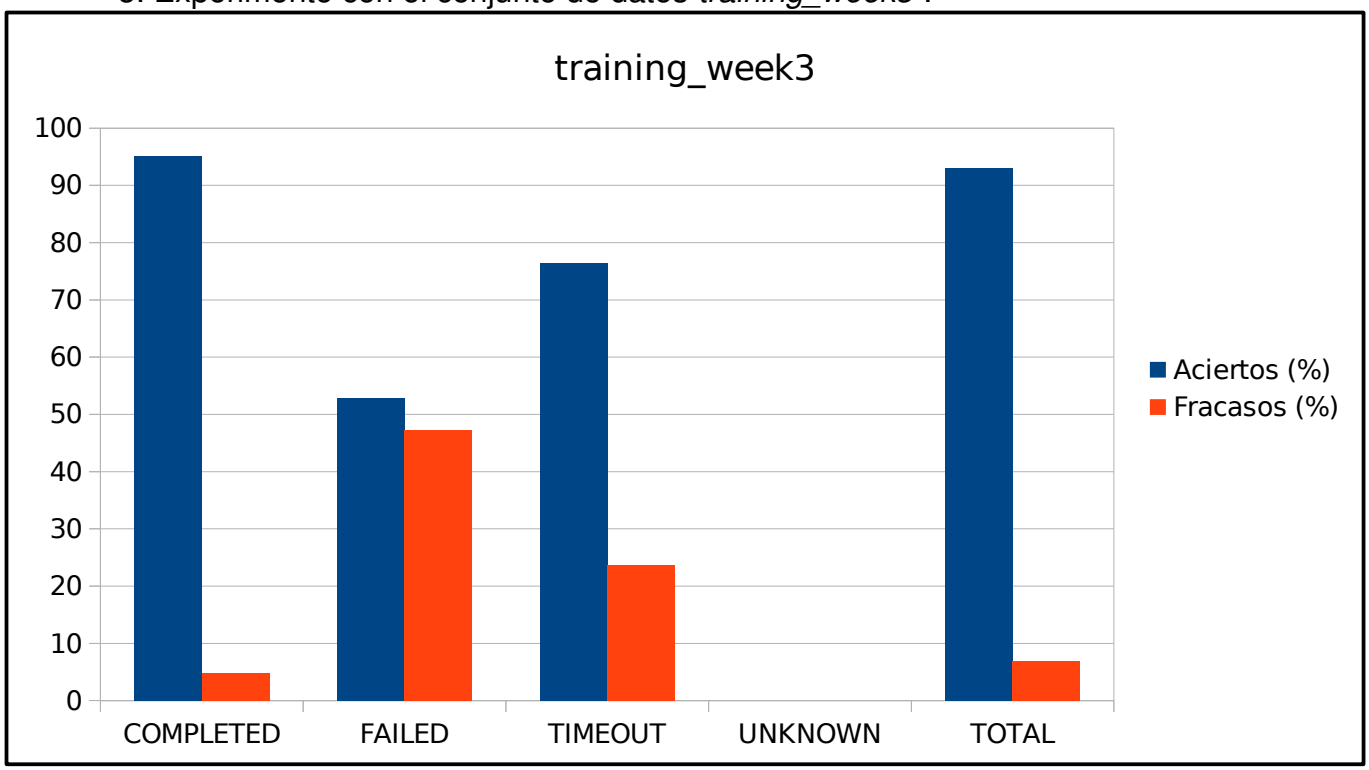
Del total del conjunto de datos, se han excluido de la clasificación el 10.75%.
Se ha clasificado incorrectamente el 10.08% de los datos clasificados.

2. Experimento con el conjunto de datos *training_week2* :



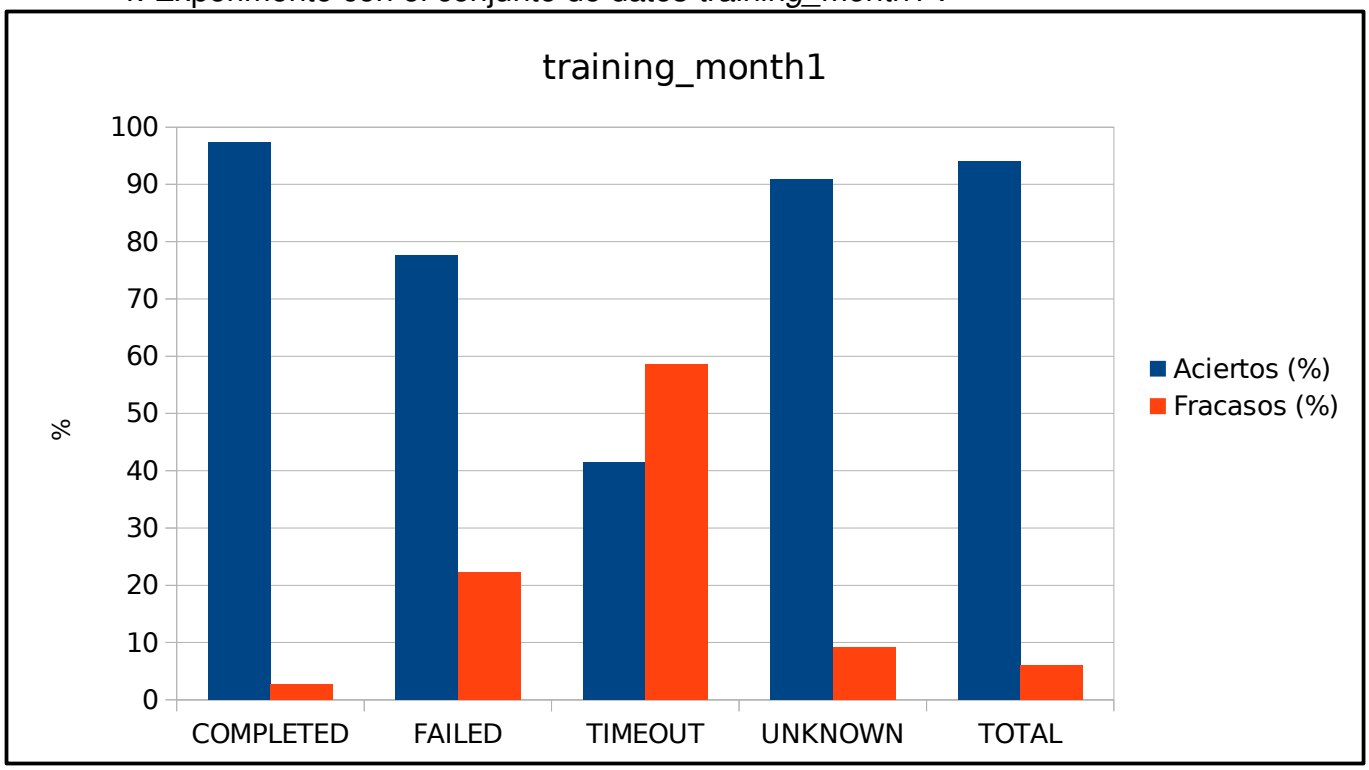
Del total del conjunto de datos, se han excluido de la clasificación el 9.12%.
Se ha clasificado incorrectamente el 6.77% de los datos clasificados.

3. Experimento con el conjunto de datos *training_week3* :



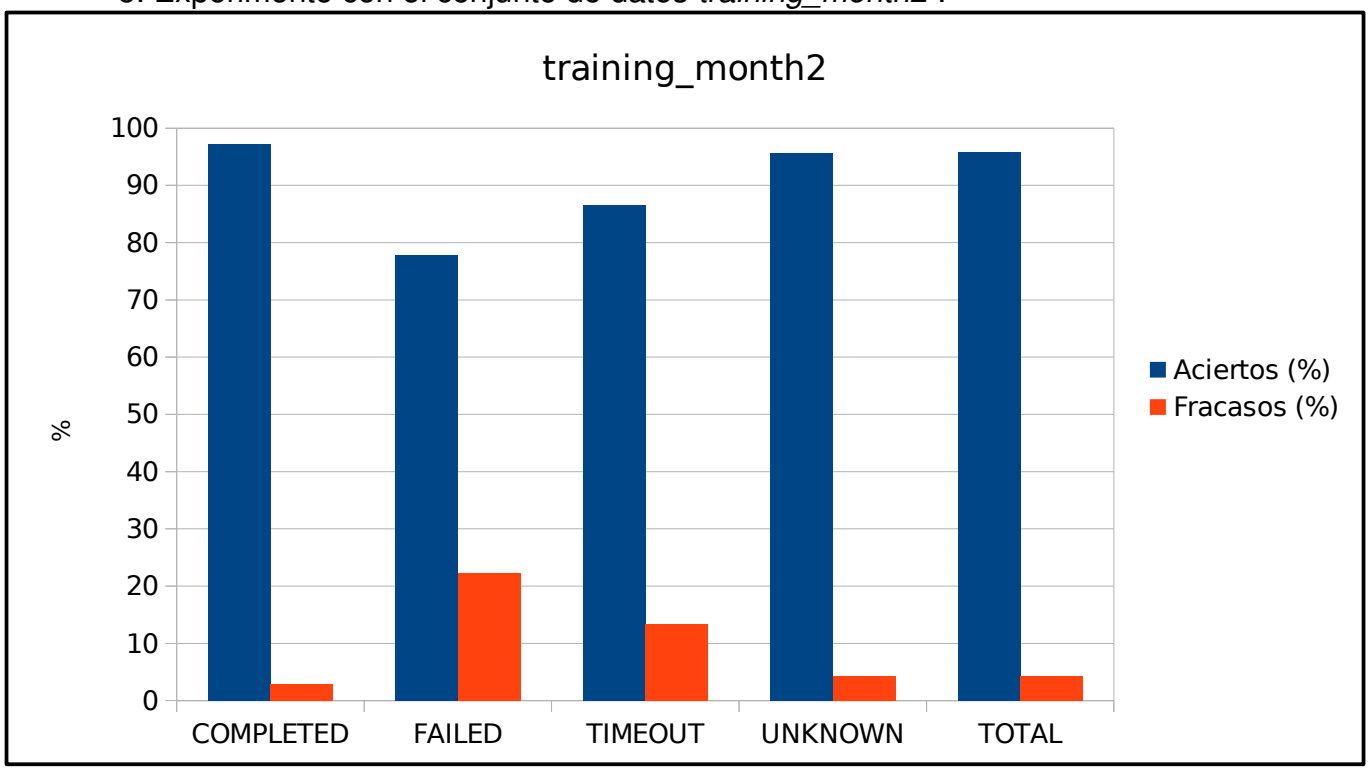
Del total del conjunto de datos, se han excluido de la clasificación el 11.37%.
Se ha clasificado incorrectamente el 6.95% de los datos clasificados.

4. Experimento con el conjunto de datos *training_month1* :



Del total del conjunto de datos, se han excluido de la clasificación el 8.63%.
Se ha clasificado incorrectamente el 5.96% de los datos clasificados.

5. Experimento con el conjunto de datos *training_month2* :



Del total del conjunto de datos, se han excluido de la clasificación el 4.48%.
Se ha clasificado incorrectamente el 4.25% de los datos clasificados.

COMPARATIVA

	training week1	training week2	training week3	training month1	training month2
Exclusión	10.75%	9.12%	11.37%	8.63%	4.48%
Error	10.08%	6.77%	6.95%	5.96%	4.25%

Puede observarse como los modelos de clasificación generados a partir de un conjunto de datos de un mes excluyen menos registros y cometen menos errores que los modelos construidos utilizando conjuntos de datos de una semana.

Así mismo, de los 5 modelos considerados en los experimentos, el peor de los casos corresponde a un modelo generado a partir de los datos de una semana (*training_week1*, con un 10.08% de tasa de errores*)* , mientras que el mejor de los casos corresponde al modelo construido utilizando los datos de un mes (*training_month2*, con un 4.25% de tasa de errores).

ANÁLISIS DE ATRIBUTOS

En la construcción de los distintos modelos, resulta de interés comprobar la significatividad de los atributos según el conjunto de datos. Como se menciona en la referencia sobre C5.0 [3], el cálculo de la significatividad de los atributos corresponde al porcentaje de registros de los datos de entrenamiento que utilizan alguna regla que incluya dicho atributo. Por ejemplo, en el caso de *training_week1* tenemos el atributo *user* con una significatividad del 98.34%, esto significa que de los 46166 registros 45399 han sido clasificados por reglas que consideran el atributo *user*.

Significatividad de los atributos con los datos *training_week1*:

```
98.93% time_elapsed
98.34% user
84.18% timelimit
64.07% partition
62.65% priority
49.53% start_hour
11.79% start_week_day
4.69% mem_req
0.34% cpus_req
0.11% account
0.07% nodes_alloc
0.03% id_qos
0.01% cpus_alloc
```

Significatividad de los atributos con los datos *training_week2*:

```
99.36% time_elapsed
96.06% user
3.19% id_qos
1.64% timelimit
0.63% mem_req
0.59% account
0.51% start_week_day
0.41% start_hour
0.17% priority
```

Significatividad de los atributos con los datos *training_week3*:

```
98.90% user
96.41% partition
88.64% time_elapsed
79.47% timelimit
69.05% account
11.11% start_hour
5.63% priority
2.41% start_week_day
1.65% cpus_alloc
0.18% nodes_alloc
0.04% id_qos
0.02% cpus_req
```

Significatividad de los atributos con los datos *training_month1*:

```
100.00% nodes_alloc
25.68% user
12.86% time_elapsed
9.99% timelimit
8.89% start_week_day
8.71% mem_req
4.56% priority
4.36% start_hour
2.33% id_qos
1.84% partition
1.63% cpus_req
0.66% account
0.63% cpus_alloc
```

Significatividad de los atributos con los datos *training_month2*:

```
100.00% time_elapsed
24.56% user
17.04% timelimit
15.10% priority
9.07% start_hour
5.06% id_qos
3.43% start_week_day
3.27% nodes_alloc
3.20% mem_req
1.03% partition
0.24% cpus_req
0.15% cpus_alloc
0.05% account
```

Puede observarse que existen ciertos atributos como *nodes_alloc*, *user* o *time_elapsed* que aparecen con una significatividad muy elevada en varios modelos, mientras que otros como *cpus_alloc* o *cpus_req* suelen tener mucho menos peso e incluso no ser considerados por algún modelo, como es el caso del construido a partir de los datos *training_week2*.

Por consiguiente, se realizan una serie de consideraciones a tener en cuenta:

La primera y más relevante es que *time_elapsed* es un atributo especial, puesto que es el único atributo que varía a lo largo del tiempo (un trabajo que no ha terminado va incrementando su valor de *time_elapsed* conforme avanza). Existe una fuerte tendencia; observable en los modelos construidos a partir de *training_week1*, *training_week2* y *training_month2*; a considerar este atributo como el más significativo. Esto implica que para una buena parte de los modelos generados, las predicciones variarán conforme se vaya avanzando en el tiempo. Por tanto, a la hora de construir el software, habrá que considerar el llevar un rastreo de las predicciones de interés para actualizarlas en distintos intervalos de tiempo. De esta manera también queda limitada la eficacia del modelo, ya que para trabajos de larga duración las primeras predicciones serán sensiblemente menos fiables que las que se realicen posteriormente(hay que tener en cuenta que los valores de de los datos de entrenamiento corresponden a trabajos ya concluidos, por tanto las predicciones derivadas de este modelo corresponderán más con la realidad cuanto más próximo esté de su terminación el trabajo).

La segunda consideración a tener en cuenta es que el grueso de los atributos tiene una significatividad altamente variable según el modelo. Por ejemplo, el atributo *partition* tiene un peso considerable en los modelos derivados de los datos *training_week3* (usado en el 96.41% de los casos) y *training_week1* (usado en el 64.07% de los casos) pero, por el contrario, en modelos derivados de otros conjuntos de datos tiene un peso mucho menor, tal es el caso de *training_week2* donde ni siquiera se considera. Lo mismo ocurre con el atributo *account* que pasa de tener un peso del 0.05% en el modelo derivado de *training_month2* a tener un peso de 69.05% en el modelo derivado de *training_week3*. Así pues, se entiende que esto se debe al estado del CPD en el momento en que fueron recolectados los datos. Como la configuración del mismo varía, y los tipos de trabajo también, en base a los proyectos en que se esté trabajando durante unos meses, que pueden ser radicalmente distintos a los que correspondan a otros, cabe esperar que ciertos atributos sean más significativos durante una época concreta pero no durante otra y, de igual manera, unos atributos que sean poco significativos en una época concreta lo sean mucho más en otras.

En cuanto a la tercera y última consideración, aunque parezca haberse observado que algunos atributos, como los relativos al uso de CPU, apenas tienen peso, partiendo de la segunda consideración (expuesta en el párrafo anterior), se respetarán; ya que no suponen un gran aumento del tamaño de los registros de datos -al menos no hasta el punto de hacerlos inasumibles, ni de afectar negativamente a su rendimiento- y, de cara al futuro, permitirán observar la evolución de estos atributos en futuros modelos (de manera similar a lo que se observa en otros campos que parecen no tener peso en algunos modelos pero ser muy significativos en otros).

CONCLUSIONES

Tras observar como se comportan los modelos construidos a partir de los datos de un período de tiempo equivalente a una semana, se concluye que no es un volumen de datos adecuado para realizar predicciones, en comparación con su alternativa.

Por otro lado, los modelos construidos a partir de los datos de un mes son mucho más prometedores y la construcción de estos -si bien no se mide, ya que no todos los meses generan el mismo volumen de datos- se realiza en un tiempo computacional aceptable.

De esta manera, se toma la decisión de utilizar dos modelos a la vez para el sistema de clasificación:

1. **Modelo con los datos de 30 días.**
   Este modelo será utilizado para realizar las clasificaciones en primer lugar. Cuando no se pueda clasificar un registro de datos utilizando este modelo, se remitirá al modelo de 90 días.

   El modelo de clasificación se renovará cada 30 días, construyendo uno nuevo y sustituyendo al viejo con este.

2. **Modelo con los datos de 90 días.**
   Este modelo será utilizado para realizar las clasificaciones en segundo lugar. Cuando un registro no pueda ser clasificado por este modelo, se considerará como inclasificable por el sistema en su estado actual.

   El modelo de clasificación se renovará cada 90 días, construyendo uno nuevo y sustituyendo el viejo por este.

   La existencia de este modelo se justifica, principalmente, en dar una respuesta a los posibles registros que el modelo de 30 días no pueda clasificar, debido a contener algún dato categórico que no se contempló en la construcción del modelo.

   No se han realizado experimentos en detalle ya que la función de este modelo es meramente auxiliar, como se ha comentado antes, para cubrir una pequeña cantidad de casos apoyando al modelo de 30 días.

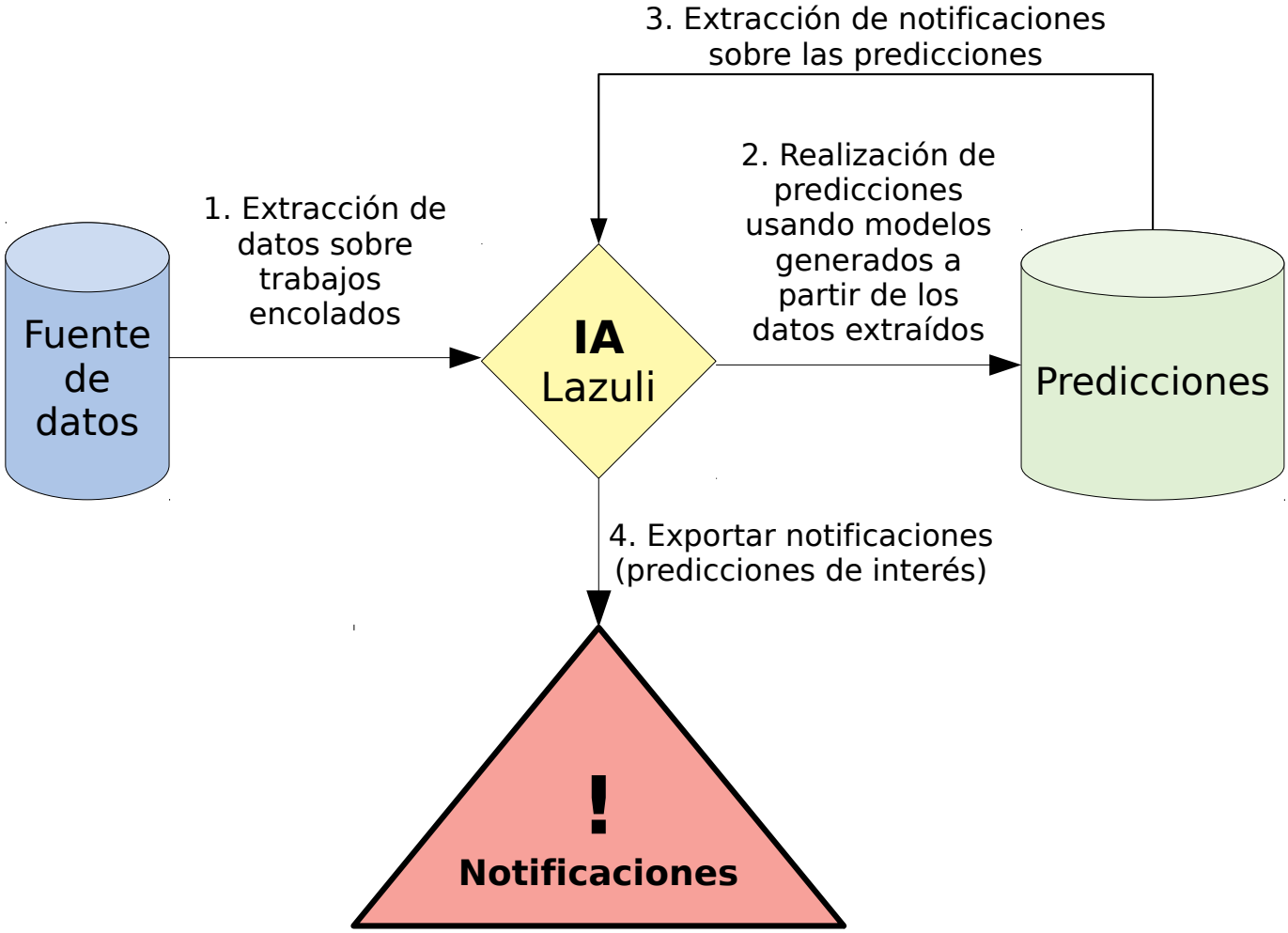
   Puesto que este modelo se construye considerando los 90 días anteriores al modelo de 30 días, se trata de un modelo que podría ser considerado antiguo y, a raíz de esto, es posible que ocurran situaciones de obsolescencia; por lo que debe considerarse que sus predicciones pueden ser poco fiables, especialmente cuando hayan ocurrido cambios significativos en la infraestructura del CPD. Por tanto, es importante tener en cuenta que este modelo, construido con datos viejos, debe ser utilizado únicamente de manera auxiliar y nunca como modelo principal. Además, su uso y generación debe ser opcional; de manera que, si así se desea, el usuario del software pueda modificar la configuración para anular dicha funcionalidad en aras de una mayor eficiencia computacional y de memoria, tanto principal como secundaria.

Por último, la decisión derivada del análisis de atributos es la de conservar todos los campos pese a observar que algunos no son especialmente significativos. Esto se debe al hecho de que algunos atributos son poco relevantes (o incluso ni se consideran) en algunos modelos, pero tienen un gran peso en otros. El ámbito de este TFG, al estar limitado a la duración del semestre académico y unos plazos marcados, permite cubrir la parte de desarrollo y las primeras evaluaciones. Sin embargo, descartar un atributo en este contexto sería algo precipitado. Por tanto, aunque se considera que algunos atributos podrían terminar siendo descartados, resultaría imprudente realizar dicha operación en un plazo de tiempo tan apresurado y, por tanto, se conservan bajo la premisa de que deben ser analizados a lo largo del tiempo y, tal vez, eventualmente terminen siendo excluidos. Dichos atributos son: *cpus_req*, *cpus_alloc*, *mem_req* e *id_qos* (han sido seleccionados debido a que en ninguno de los modelos han logrado una significatividad de al menos el 10%).

2.3. Software final

ESQUEMAS DE FUNCIONAMIENTO

Esquema general

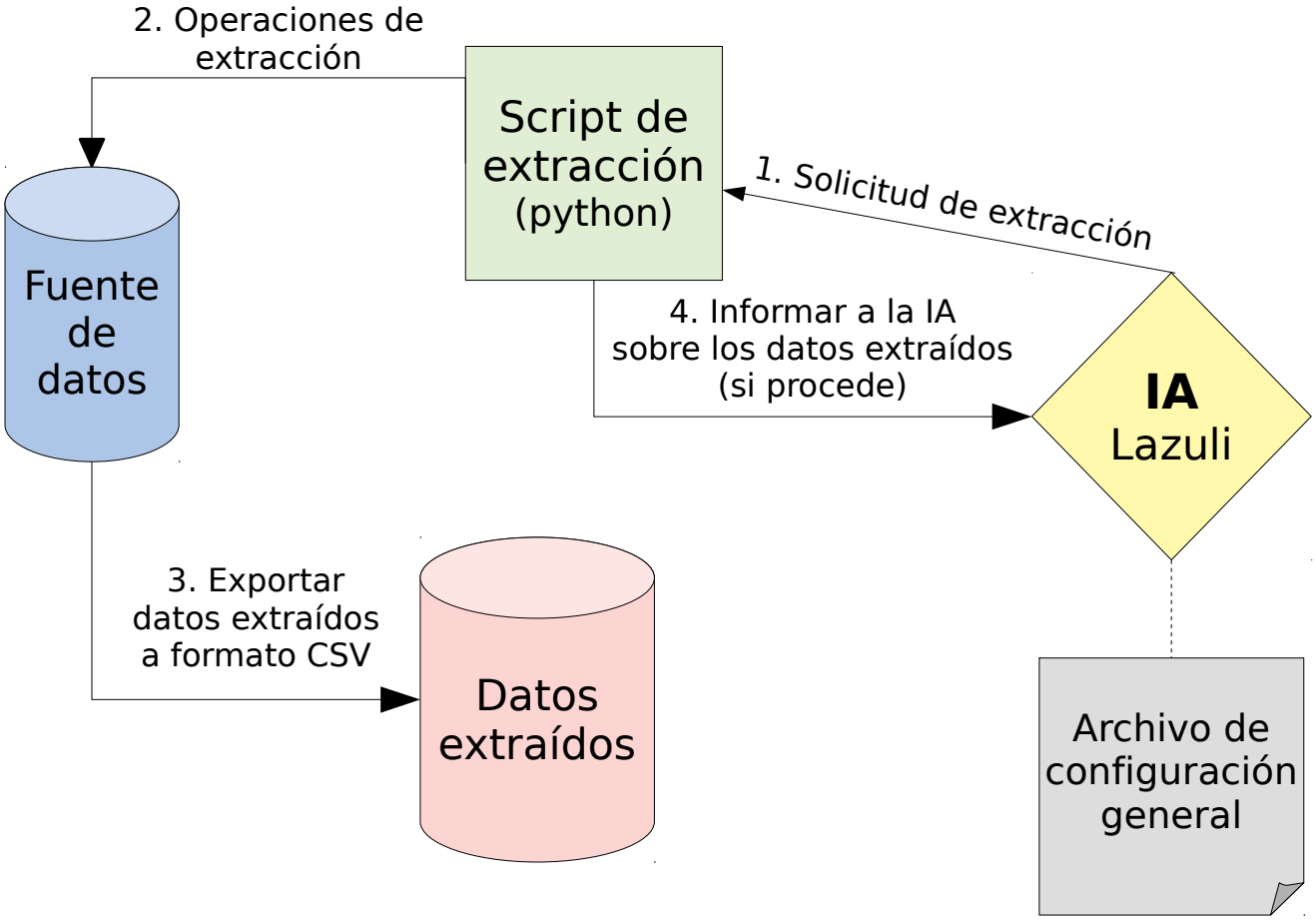


El concepto general del software final consiste en extraer de la fuente de datos del CPD los registros correspondientes al encolamiento de los trabajos asociados a un rango de tiempo determinado (los últimos 30 días para el modelo principal y los 90 anteriores a estos 30 para el secundario, con la configuración utilizada para este TFG) y generar los modelos correspondientes.

Una vez generados los modelos, se extraen de la fuente de datos aquellos registros que correspondan al intervalo de tiempo configurado desde la última extracción reconocida y se realizan las predicciones oportunas.

Sobre las predicciones realizadas, se aplican los filtros definidos en el archivo de configuración para seleccionar aquellas de interés y exportarlas como notificaciones.

Esquema de extracción de datos



Cuando se realizan operaciones de extracción de datos, tanto si son datos para generar modelos como si son datos sobre los que realizar predicciones, el software lanza el script en *bash* que corresponda al tipo de operación para que utilice el script de extracción en *python* que se comunica con la fuente de datos y genere una serie de registros en formato CSV que correspondan a la información solicitada.

Las rutas a los directorios y los scripts a utilizar están especificadas en el archivo de configuración general. Sobre esto no se especificará nada más aquí, pues en la siguiente sección (ARCHIVOS DE CONFIGURACIÓN) se comenta en detalle el funcionamiento de estos archivos. En su lugar, se presentan a continuación los 3 scripts en bash utilizados para asistir en la extracción de datos facilitando la comunicación entre la IA y el script de extracción en *python*:

short_collection_script:

```
#!/bin/bash
# $1 -> Directory where the file will be exported

if [ $# -lt 1 ]; then
cat << EOF
No directory was specified.
EOF
        exit 1
fi


now=`date +%s`

start_epoch=$(($now-2592000))
end_epoch=$now

start_date=`date -d @$start_epoch +"%Y-%m-%dT%H:%M:%S"`
end_date=`date -d @$end_epoch +"%Y-%m-%dT%H:%M:%S"`

file_name='SD_FROM_'$start_date'_TO_'$end_date'.csv'

python export_metrics/export_metrics.py -out $1/$file_name -started-after-ts $start_epoch -
started-before-ts $end_epoch
data/fix.sh $1/$file_name $1/$file_name'.tmp'
grep -v RUNNING $1/$file_name'.tmp' > $1/$file_name

echo $1/$file_name
```

Este script recibe como primer y único parámetro el directorio (tomado del archivo de configuración) donde exportar los datos recolectados, que corresponden a los 30 días anteriores al momento presente.

El nombre del archivo en que se exportarán los datos comienza por el prefijo SD (*ShortData*) y a continuación indica el intervalo temporal desde donde (*From)* hasta cuando (T*o*) que corresponde a los trabajos extraídos. Dicho nombre (considerando la ruta completa) es presentado como salida del script cuando la ejecución concluye con normalidad.

long_collection_script:

```
#!/bin/bash
# $1 -> Directory where the file will be exported

if [ $# -lt 1 ]; then
cat << EOF
No directory was specified.
EOF
        exit 1
fi


now=`date +%s`

start_epoch=$(($now-10368000))
end_epoch=$(($now-2592000))

start_date=`date -d @$start_epoch +"%Y-%m-%dT%H:%M:%S"`
end_date=`date -d @$end_epoch +"%Y-%m-%dT%H:%M:%S"`

file_name='LD_FROM_'$start_date'_TO_'$end_date'.csv'

python export_metrics/export_metrics.py -out $1/$file_name -started-after-ts $start_epoch -
started-before-ts $end_epoch
data/fix.sh $1/$file_name $1/$file_name'.tmp'
grep -v RUNNING $1/$file_name'.tmp' > $1/$file_name

echo $1/$file_name
```

Este script funciona exactamente igual que *short_collection_script* con la diferencia de que en lugar de considerar los últimos 30 días, considera los trabajos desde hace 120 días hasta hace 30 días (intervalo de 90 días).

El nombre del archivo cumple el mismo formato que *short_collection_script* pero cambiando el prefijo por LD (*LongData*). Dicho nombre también es presentado como salida del script cuando la ejecución concluye con normalidad.

data_to_predict_collection_script.sh:

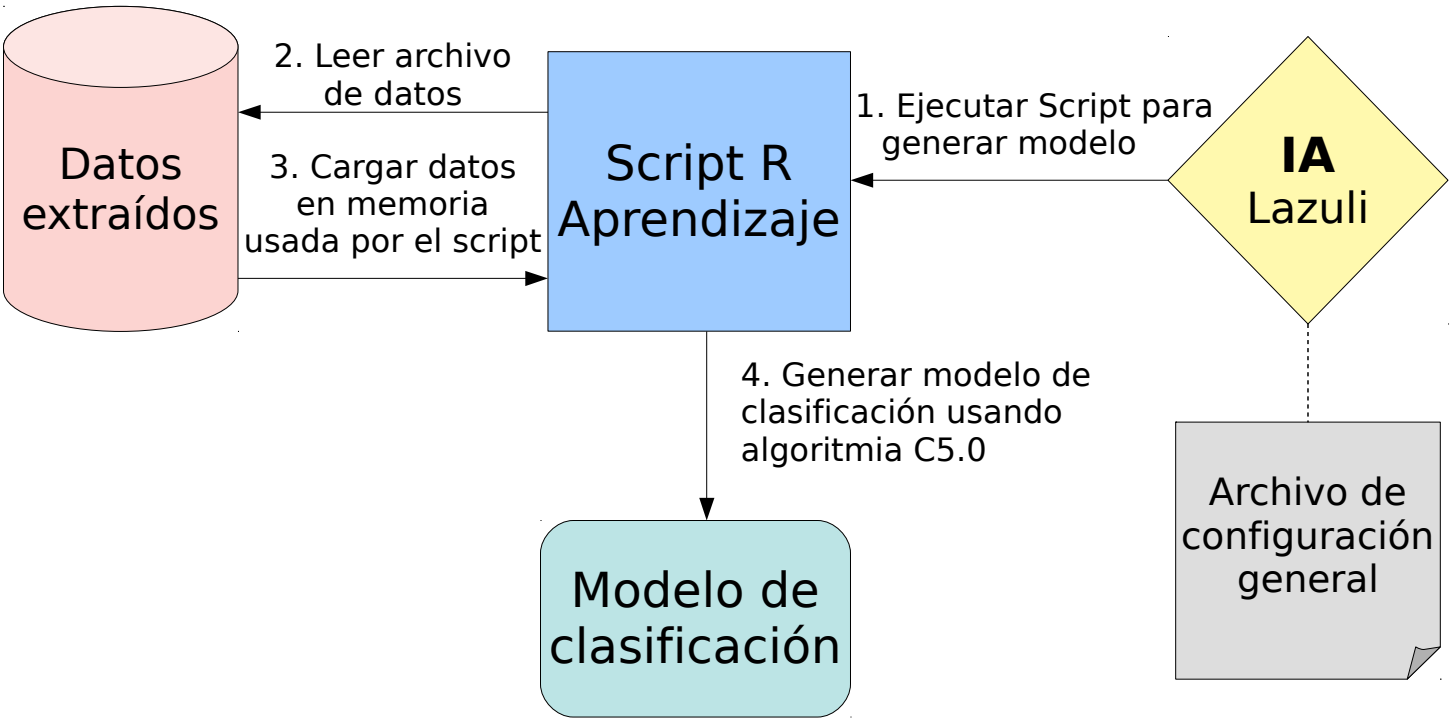
```
#!/bin/bash
# $1 -> Directory where the file will be exported
# $2 -> Collection start epoch
# $3 -> Collection end epoch
# $4 -> Path to pool of notifications that must be included in the query

if [ $# -lt 1 ]; then
cat << EOF
No directory was specified.
EOF
        exit 1
fi


now=`date +%s`

start_epoch=$2
end_epoch=$3

start_date=`date -d @$start_epoch +"%Y-%m-%dT%H:%M:%S"`
end_date=`date -d @$end_epoch +"%Y-%m-%dT%H:%M:%S"`

file_name='DATA_FROM_'$start_date'_TO_'$end_date'.csv'

if [ -f $4 ]; then
        python    export_metrics/export_metrics.py    -out    $1/$file_name    -started-after-ts
$start_epoch -started-before-ts $end_epoch -running-only -extra-queries $4
else
        python    export_metrics/export_metrics.py    -out    $1/$file_name    -started-after-ts
$start_epoch -started-before-ts $end_epoch -running-only
fi
data/fix.sh $1/$file_name $1/$file_name'.tmp'
mv $1/$file_name'.tmp' $1/$file_name

echo $1/$file_name
```

Este script es utilizado para comunicar el software con el script de extracción en *python* con la finalidad de extraer datos para realizar predicciones sobre los trabajos a los que corresponden.

El primer parámetro de este script corresponde al directorio donde se exportarán los datos extraídos. El segundo parámetro es la fecha (en formato *epoch* o Tiempo POSIX/Unix [20]) de comienzo del intervalo de tiempo de los datos a extraer y el tercer parámetro el final. El cuarto parámetro es la ruta al *pool* de notificaciones, un archivo que contiene un id de trabajo en cada linea por cada trabajo que se considere de interés para el sistema de notificaciones, dichos trabajos serán predecidos mientras sigan figurando en la lista, debido a la problemática con el atributo *time_elapsed* que se ha explicado en el apartado de Construcción *y elaboración de modelos* (2.2), más concretamente en la sección *Análisis de atributos*. Además, si el cuarto parámetro no corresponde a un archivo existente se invocará al script de extracción de datos sin considerar la *pool* de predicciones de interés.

Finalmente, los datos extraídos son exportados a un archivo en el directorio correspondiente de manera análoga a los otros dos scripts que asisten en la extracción de datos y sin utilizar ningún prefijo especial.

Esquema de generación de modelos



Los modelos de clasificación se construyen a partir de la recolección de datos para tal fin. Una vez esta operación (especificada en el apartado anterior de esta sección) ha sido llevada a cabo con éxito, la IA conoce el archivo que contiene los registros extraídos y lanza sobre este conjunto de datos el script en *R* que los analiza para generar un modelo de clasificación en la ruta definida en el archivo de configuración.

Es importante tener en cuenta que, debido al uso que se hace del modelo, antes de actualizar uno ya existente será necesario obtener el acceso exclusivo al mismo para realizar la operación de lectura. De esta manera, se garantiza que no se modifique un modelo que pudiese estar siendo utilizado para realizar alguna predicción. El acceso a este recurso compartido se realiza mediante las funcionalidades que aporta la clase *ModelSynchronizer*.

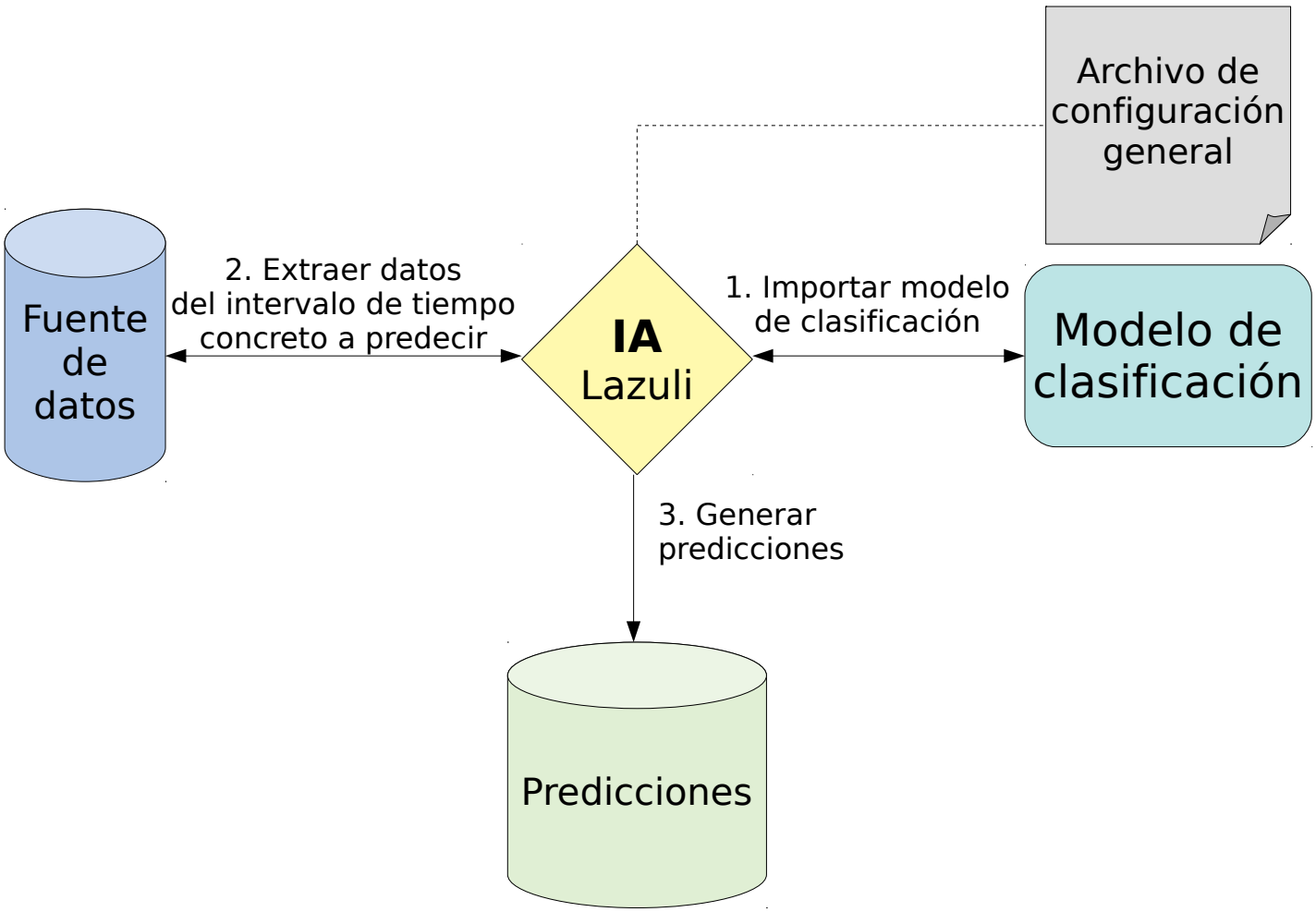
Así mismo, el script en R utilizado es una versión adaptada del anteriormente expuesto, que excluye aquellas líneas que eran de interés en la fase de desarrollo pero carecen de utilidad en la implementación. Dicho script es el que sigue:

```
#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)

# args[1] -> Path to the training CSV file
# args[2] -> Path where the generated model will be exported

# C5.0 for decission trees and rule-based models

# Libraries
library(C50)

# Data importing (without id field)
training_data = read.csv(args[1], header = TRUE)[,-6][,-11:-12][,-12:-13][,-1]

# Build model
model <- C5.0(x = training_data[,-length(training_data)], y = training_data$state, rules
= TRUE, trials=1)
write(model$rules, file=args[2])
```

Esta versión del script recibe dos parámetros, el primero que es la ruta al archivo CSV utilizado para generar el modelo y el segundo que es la ruta a donde se habrá de exportar el modelo.

Tras recibir los parámetros, se carga la librería *C50*, se importan los datos, se construye el modelo y se exporta al archivo correspondiente.

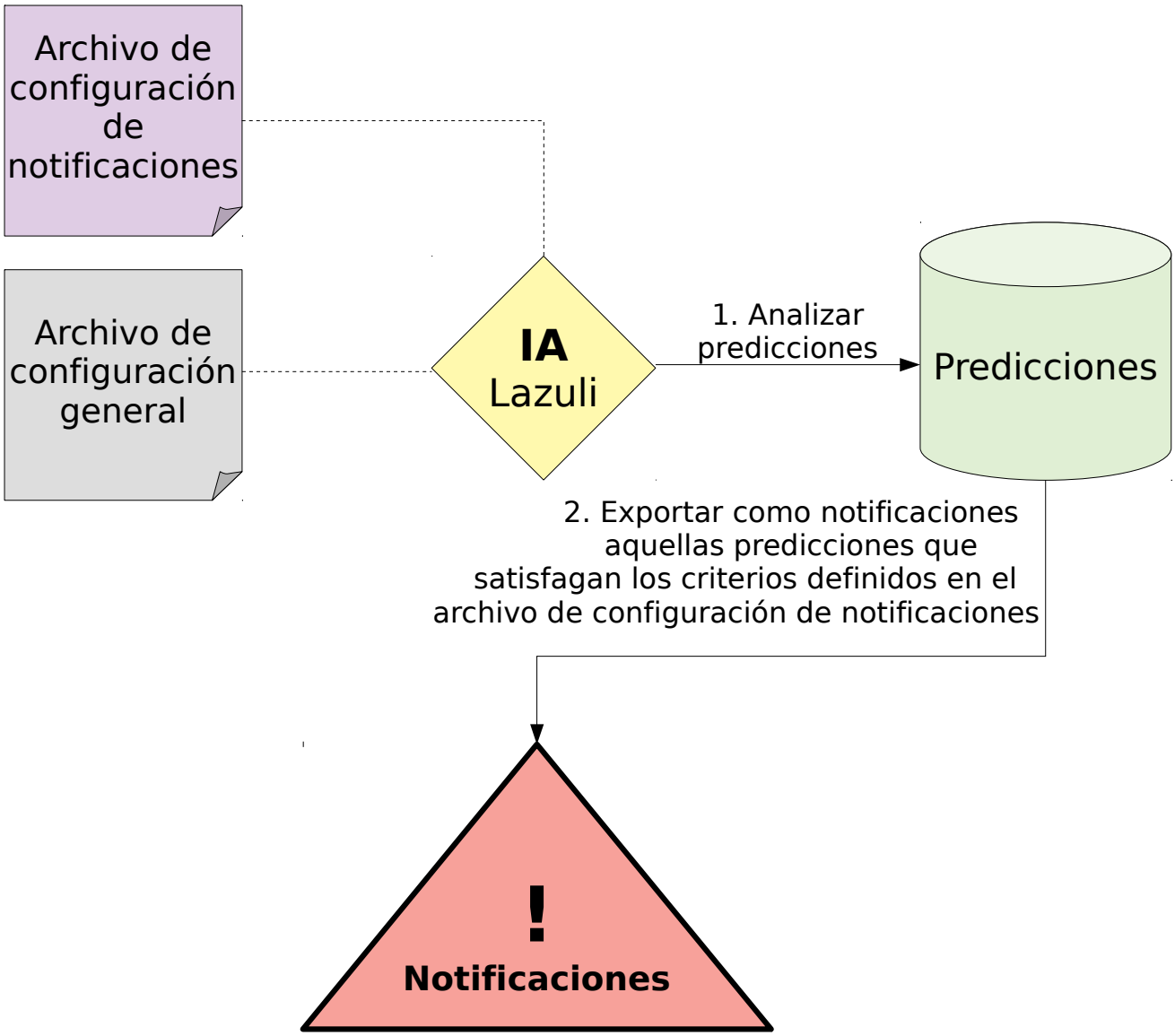
Esquema de predicción



Cada cierto intervalo de tiempo, especificado en el archivo de configuración general, la IA recolecta los datos relativos a trabajos encolados desde la última recolección hasta el presente. Sobre estos datos, se aplica el modelo de clasificación y, del resultado de esta aplicación, surgen las predicciones que se exportan al directorio oportuno, nombrando los archivos en base a la fecha en que fueron generados.

Para utilizar el modelo de clasificación, el software parsea el archivo de reglas y carga en memoria el modelo en un sistema de clases que permite su aplicación. Es necesario consultar los detalles de funcionamiento del conjunto de reglas generado mediante la algoritmia *C5.0* [21] para entender el *parser* y la aplicación del modelo sobre los conjuntos de datos a clasificar.

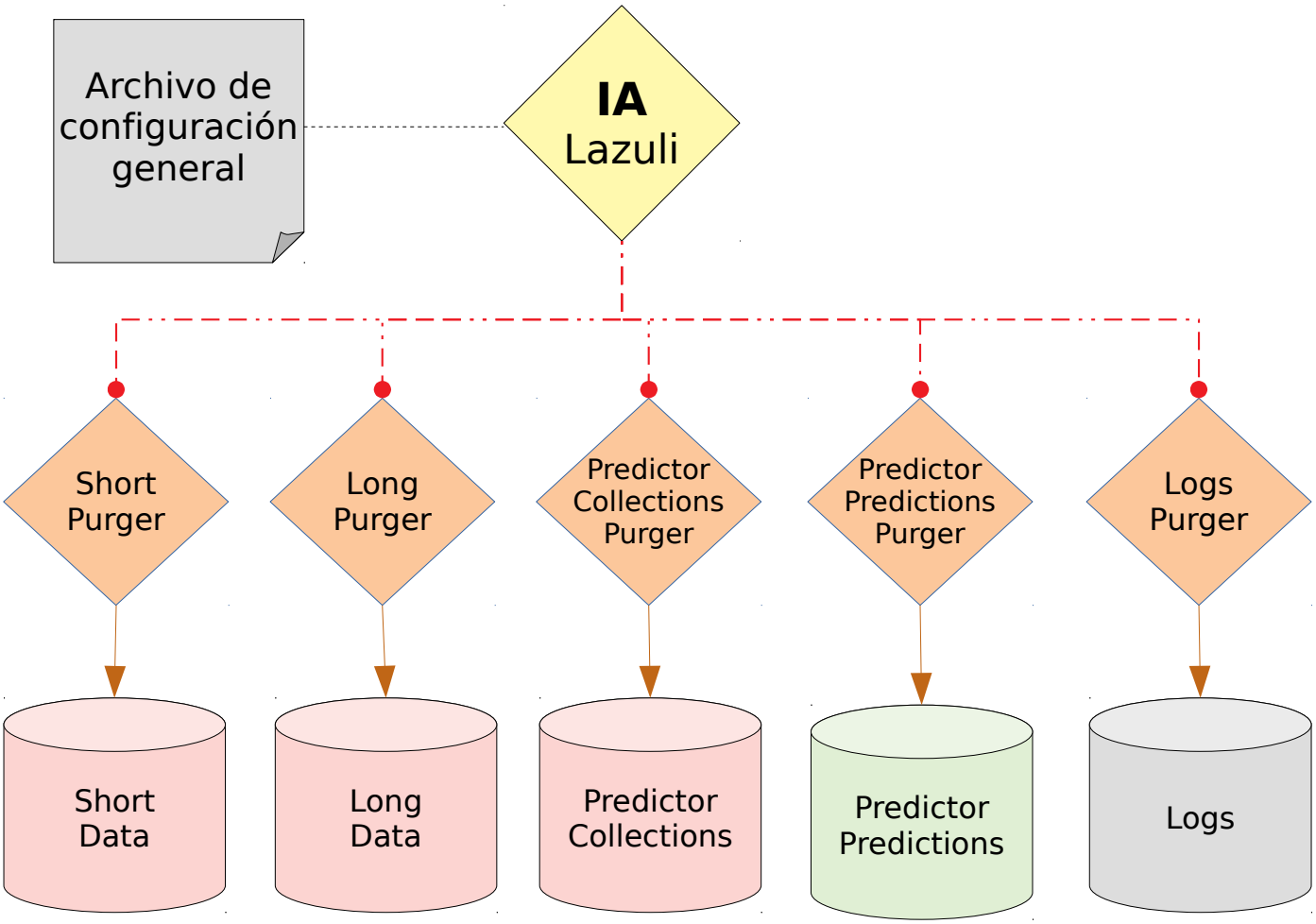
Esquema de notificación



En el archivo de configuración de notificaciones (que se explicará en la siguiente sección) se definen unos filtros determinados que, cuando sean cumplidos por alguna predicción, implican que dicho registro sea considerado para exportar como notificación.

Las notificaciones son, en realidad, un subconjunto contenido en el conjunto del total de las predicciones.

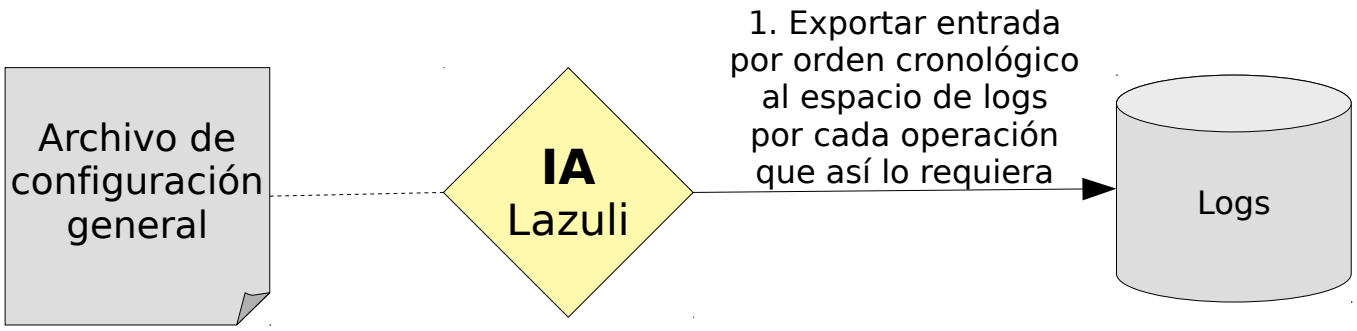
Esquema de control de archivos



Cuando el software funciona como un servicio (el modo de funcionamiento de la IA), genera datos que se guardan en el almacenamiento persistente. Con el fin de evitar que se guarden datos obsoletos que ya no son de interés, de manera que ocupen un espacio de almacenamiento que podría destinarse a otras funciones, se ha diseñado un sistema de purga de archivos.

El sistema de purgas genera un hilo independiente por cada bloque de almacenamiento (*short data*, *long data*, *predictor collections*, *predictor predictions* y *logs*), dicho hilo se configura en base al criterio de tiempo de vida fijado en el archivo de configuración general (que se explicará en la siguiente sección). Todos aquellos archivos que hayan superado la antigüedad especificada para el tipo de datos correspondiente (considerando siempre el tiempo desde la última modificación), son eliminados del sistema de almacenamiento persistente.

Esquema de logs



La inteligencia artificial realiza operaciones de diversa índole, muchas de las cuales son susceptibles de dar lugar a errores o de ser de gran relevancia para el sistema, por lo cual conocer su estado y terminación (ya sea esta exitosa o errada) se convierte en información de interés.

Para dejar constancia este tipo de eventos, se ha programado un sencillo sistema de logs que exporta al almacenamiento persistente los mensajes considerados de interés para el sistema.

Los logs se clasifican en directorios que corresponden en primer nivel al año y en segundo al mes, dentro del directorio de cada mes los archivos se nombran considerando el día al que corresponden las notificaciones y, los mensajes emitidos, comienzan por la hora en que fueron exportados.

ARCHIVOS DE CONFIGURACIÓN

Para especificar las distintas directrices a seguir por el software, se utilizan archivos de configuración a través de cuyos parámetros se establecen los tiempos, directorios y criterios de uso de la aplicación.

Existen dos archivos de configuración, el primero y más importante es el archivo de configuración general en el cual se parametriza el funcionamiento de la inteligencia artificial. El segundo archivo corresponde al sistema de notificaciones y se utiliza para indicar los filtros que definen aquellas predicciones de interés para ser consideradas por el sistema de notificaciones.

CONFIGURACIÓN GENERAL

La configuración general consta de los siguientes campos:

NotifierConfiguration

Especifica la ruta al archivo de configuración de las notificaciones. Si el archivo no existe, se construirá una versión por defecto del mismo en dicha ruta.

Ejemplo:

```
NotifierConfiguration=data/notifier/config.cfg
```

NotifierPool

Especifica la ruta al archivo que contiene, en cada línea, un id que identifica los trabajos que se están prediciendo y deben volver a ser consultados hasta que terminen o dejen de satisfacer los criterios que definen las predicciones a considerar como notificación.

Ejemplo:

```
NotifierPool=data/notifier/pool
```

NotificationsPath

Especifica la ruta al archivo donde se exportarán las notificaciones.

Ejemplo:

```
NotificationsPath=data/notifier/notifications.csv
```

PredictorPath

Especifica la ruta al directorio base de las predicciones. Actualmente no se está utilizando en el software, pero se contempla el campo tanto por motivos de retrocompatibilidad como de uso en futuras versiones. No es necesario que figure en el archivo de configuración, puede omitirse.

Ejemplo:

```
PredictorPath=data/predictor/
```

PredictorScriptPath

Especifica la ruta al script utilizado para recolectar los datos cuyo estado se ha de predecir.

Ejemplo:

```
PredictorScriptPath=data/predictor/
data_to_predict_collection_script.sh
```

PredictorCollectionPath

Especifica la ruta al directorio donde se guardarán los datos recolectados antes de ser clasificados, es decir, de realizar las predicciones.

Ejemplo:

PredictorCollectionPath=data/predictor/collections

PredictorPredictionsPath

Especifica la ruta al directorio donde se exportarán los archivos que contengan las predicciones realizadas.

Ejemplo:

PredictorPredictionsPath=data/predictor/predictions

PredictorInterval

Especifica cada cuanto tiempo (en segundos) se llevará a cabo la recolección de datos a predecir.

Ejemplo:

PredictorInterval=90

PredictorCollectionsTimeToLive

Especifica el tiempo de vida (en segundos) de los datos a predecir recolectados. Aquellos archivos de datos cuya antigüedad supere este parámetro, serán purgados. Si el valor es 0, no se realizará ninguna purga.

Ejemplo:

PredictorCollectionsTimeToLive=300

PredictorPredictionsTimeToLive

Especifica el tiempo de vida (en segundos) de las predicciones. Aquellos archivos de predicciones cuya antigüedad supere este parámetro, serán purgados. Si el valor es 0, no se realizará ninguna purga.

Ejemplo:

PredictorPredictionsTimeToLive=600

PredictorLastCollectionEpoch

Especifica la fecha (en formato *epoch*) de la última recolección de datos a predecir realizada. En caso de no haberse realizado la primera recolección todavía, debe indicarse igualmente este parámetro pues en la siguiente recolección, aunque sea la primera, se considerarán únicamente aquellos datos comprendidos en el intervalo temporal entre esta fecha y el presente; descartando todos los anteriores.

Ejemplo:

PredictorLastCollectionEpoch=1524482222

LogsPath

Especifica la ruta al directorio donde se exportarán los logs, clasificándolos en subdirectorios por año y mes y generando un archivo distinto para cada día.

Ejemplo:

LogsPath=logs

LogsTimeToLive

Especifica el tiempo de vida (en segundos) de los logs. Aquellos archivos que correspondan a logs y tengan una antigüedad que supere este parámetro, serán purgados. Si el valor es 0, no se realizará ninguna purga.

Ejemplo:

```
LogsTimeToLive=0
```

ShortDataPath

Especifica la ruta al directorio donde se recolectarán los datos a utilizar para construir modelos de predicción cortos, que son los principales, es decir, los primeros que se utilizarán para tratar de predecir.

Ejemplo:

```
ShortDataPath=data/short/collections
```

ShortCollectionScript

Especifica la ruta al script utilizado para recolectar los datos que se utilizarán para construir modelos cortos/principales.

Ejemplo:

```
ShortCollectionScript=data/short/short_collection_script.sh
```

ShortModelScript

Especifica la ruta al script utilizado para generar el modelo de clasificación corto/principal con los datos de la recolección (por cada recolección se genera un modelo, no se utilizan las recolecciones anteriores).

Ejemplo:

```
ShortModelScript=data/c50.R
```

ShortModelPath

Especifica la ruta donde se exportará el modelo corto/principal generado, que también será utilizado para realizar las predicciones oportunas.

Ejemplo:

```
ShortModelPath=data/short/short.rules
```

ShortCollectionInterval

Especifica cada cuanto tiempo (en segundos) se realiza la recolección de datos para construir un nuevo modelo corto/principal.

Ejemplo:

```
ShortCollectionInterval=6000
```

ShortDataTimeToLive

Especifica el tiempo de vida (en segundos) de los archivos de datos usados por los modelos cortos/principales. Aquellos archivos cuya antigüedad supere este parámetro, serán purgados. Si el valor es 0, no se realizará ninguna purga.

Ejemplo:

```
ShortDataTimeToLive=300
```

LongDataEnabled

Especifica, mediante los valores *true*/*false*, si el uso del modelo de clasificación secundario está activo o no. Cuando no esté activo, no se recolectarán datos ni se construirán modelos de predicción largos/secundarios.

Ejemplo:

```
LongDataEnabled=true
```

LongDataPath

Especifica la ruta al directorio donde se recolectarán los datos a utilizar para construir modelos de predicción largos, que son los secundarios, es decir, los que se utilizarán para tratar de predecir cuando el principal no cubra los casos.

Ejemplo:

```
LongDataPath=data/long/collections
```

LongCollectionScript

Especifica la ruta al script utilizado para recolectar los datos que se utilizarán para construir modelos largos/secundarios.

Ejemplo:

```
LongCollectionScript=data/long/long_collection_script.sh
```

LongModelScript

Especifica la ruta al script utilizado para generar el modelo de clasificación largo/secundario con los datos de la recolección (por cada recolección se genera un modelo, no se utilizan las recolecciones anteriores).

Ejemplo:

```
LongModelScript=data/c50.R
```

LongModelPath

Especifica la ruta donde se exportará el modelo largo/secundario generado, que también será utilizado para realizar las predicciones oportunas.

Ejemplo:

```
LongModelPath=data/long/long.rules
```

LongCollectionInterval

Especifica cada cuanto tiempo (en segundos) se realiza la recolección de datos para construir un nuevo modelo largo/secundario.

Ejemplo:

```
LongCollectionInterval=12000
```

LongDataTimeToLive

Especifica el tiempo de vida (en segundos) de los archivos de datos usados por los modelos largos/secundarios. Aquellos archivos cuya antigüedad supere este parámetro, serán purgados. Si el valor es 0, no se realizará ninguna purga.

Ejemplo:

```
LongDataTimeToLive=600
```

El contenido del archivo de configuración general utilizado en el primer despliegue del software es el que sigue:

```
PredictorCollectionsTimeToLive=300
PredictorInterval=3600
ShortModelScript=data/c50.R
LongDataPath=data/long/collections
LongDataEnabled=true
ShortCollectionInterval=2592000
ShortCollectionScript=data/short/short_collection_script.sh
ShortModelPath=data/short/short.rules
LongCollectionInterval=7776000
PredictorPath=data/predictor/
LongCollectionScript=data/long/long_collection_script.sh
NotifierConfiguration=data/notifier/config.cfg
LongDataTimeToLive=10368000
ShortDataPath=data/short/collections
PredictorScriptPath=data/predictor/
data_to_predict_collection_script.sh
NotificationsPath=data/notifier/notifications.csv
PredictorLastCollectionEpoch=1524650420
LongModelPath=data/long/long.rules
LogsPath=logs
LongModelScript=data/c50.R
PredictorPredictionsPath=data/predictor/predictions
ShortDataTimeToLive=5184000
PredictorPredictionsTimeToLive=7200
LogsTimeToLive=15552000
PredictorCollectionPath=data/predictor/collections
NotifierPool=data/notifier/pool
```

CONFIGURACIÓN DE NOTIFICACIONES

La configuración de notificaciones permite ignorar un criterio fijando el valor de este a *null* y consta de los siguientes campos:

StateFilter

Especifica la lista de estados predichos que se consideran de interés. Cualquier registro cuyo estado predicho no figure en esta lista, será descartado.

Ejemplo:

```
StateFilter=[FAILED,TIMEOUT]
```

AccountFilter

Especifica la lista de cuentas que se consideran de interés. Cualquier registro cuyo cuenta no figure en esta lista, será descartado.

Ejemplo:

```
AccountFilter=[other]
```

UserFilter

Especifica la lista de usuarios que se consideran de interés. Cualquier registro cuyo usuario no figure en esta lista, será descartado.

Ejemplo:

```
UserFilter=null
```

PartitionFilter

Especifica la lista de particiones que se consideran de interés. Cualquier registro cuya partición no figure en esta lista, será descartado.

Ejemplo:

```
PartitionFilter=[thin-nodes,shared]
```

NodeFilter

Especifica la lista de nodos que se consideran de interés. Cualquier registro que no contenga alguno de estos nodos, será descartado.

Ejemplo:

```
NodeFilter=[c1114,c1217,c1222,c6901,c7327]
```

MinMemReq

Especifica el valor mínimo de memoria solicitada que debe tener un registro para ser considerado de interés.

Ejemplo:

```
MinMemReq=512
```

MaxMemReq

Especifica el valor máximo de memoria solicitada que debe tener un registro para ser considerado de interés.

Ejemplo:

```
MaxMemReq=32000
```

MinNodesAlloc

Especifica el valor mínimo de nodos utilizados que debe tener un registro para ser considerado de interés.

Ejemplo:

```
MinNodesAlloc=1
```

MaxNodesAlloc

Especifica el valor máximo de nodos utilizados que debe tener un registro para ser considerado de interés.

Ejemplo:

```
MaxNodesAlloc=1
```

MinCpusAlloc

Especifica el valor mínimo de CPU utilizadas que debe tener un registro para ser considerado de interés.

Ejemplo:

MinCpusAlloc=1

MaxCpusAlloc

Especifica el valor máximo de CPU utilizadas que debe tener un registro para ser considerado de interés.

Ejemplo:

MaxCpusAlloc=4

MinCpusReq

Especifica el valor mínimo de CPU solicitadas que debe tener un registro para ser considerado de interés.

Ejemplo:

MinCpusReq=1

MaxCpusReq

Especifica el valor máximo de CPU solicitadas que debe tener un registro para ser considerado de interés.

Ejemplo:

MaxCpusReq=4

MinElapsedTime

Especifica el tiempo transcurrido mínimo (en segundos) que debe tener un registro para ser considerado de interés.

Ejemplo:

MinElapsedTime=1800

MaxElapsedTime

Especifica el tiempo transcurrido máximo (en segundos) que debe tener un registro para ser considerado de interés.

Ejemplo:

MaxElapsedTime=null

MinStartTime

Especifica el la marca temporal de comienzo mínima que debe tener un registro para ser considerado de interés.

Ejemplo:

MinStartTime=1517587151

MaxStartTime

Especifica el la marca temporal de comienzo máxima que debe tener un registro para ser considerado de interés.

Ejemplo:

MaxStartTime=1517643243

Un ejemplo de contenido válido para el archivo de configuración de notificaciones que podría utilizarse en un despliegue del software es el que sigue:

```
StateFilter=[FAILED,TIMEOUT]
MinMemReq=null
StartWeekDayFilter=null
MaxNodesAlloc=null
MinNodesAlloc=null
MinStartTime=null
MinCpusAlloc=null
MaxMemReq=null
MinCpusReq=null
MinAccuracy=0.90
MaxCpusAlloc=null
UserFilter=null
NodeFilter=null
MaxCpusReq=null
MinElapsedTime=null
AccountFilter=null
PartitionFilter=null
MaxStartTime=null
MaxElapsedTime=null
```

CONSULTAS

El software final, además del modo de funcionamiento como servicio de inteligencia artificial, puede ser utilizado para ejecutar consultas sobre los archivos de datos en formato CSV. Dichas consultas son utilizadas también por la IA para aplicar filtros a las predicciones y considerar aquellas que cumplen los criterios para ser consideradas notificaciones.

Con el fin de entender mejor el funcionamiento de las consultas, se presentan a continuación unos ejemplos de uso sobre el conjunto de datos *training_month2*, considerando únicamente 1000 registros elegidos al azar para facilitar la comprensión:

1. Obtener aquellos registros cuyo estado sea *FAILED* o *TIMEOUT*, cuya cuenta asociada sea *other*, que hayan solicitado como máximo 1 CPU y cuya ejecución haya durado como mínimo 10 minutos (600 segundos)

INSTRUCCIÓN

```
java -jar lazuli.jar -q data.csv STATE=[FAILED,TIMEOUT] ACCOUNT=[other] MAX_CPUS_REQ=1 MIN_ELAPSED_TIME=600
```

SALIDA

```
id_job, priority, account, user, nodes_alloc, nodelist, cpus_req, cpus_alloc, mem_req, id_qos, partition, time_start, time_end, timelimit, time_submit, time_eligible, time_suspended, time_elapsed, start_week_day, start_hour, state
877480, 609280, other, ulctidrc, 1, [c1225], 1, 1, 2147485696, 52, shared, 1517514401, 1517874428, 6000, 1517404296, 1517404298, 0, 360027, thursday, 20, TIMEOUT
1011258, 3152895, other, uscmgftr, 1, [c6601], 1, 1, 2147488768, 43, thin-interactive, 1519060810, 1519065591, 480, 1519060755, 1519060755, 0, 4781, monday, 18, FAILED
887282, 3137444, other, uvibgmga, 1, [c6601], 1, 1, 2147488768, 43, thin-interactive, 1517659917, 1517688742, 480, 1517659911, 1517659911, 0, 28825, saturday, 13, TIMEOUT
869499, 509916, other, ulctidrc, 1, [c1134], 1, 1, 2147485696, 52, shared, 1517427767, 1517787783, 6000, 1517404296, 1517404298, 0, 360016, wednesday, 20, TIMEOUT
868758, 495372, other, ulctidrc, 1, [c1107], 1, 1, 2147485696, 52, shared, 1517415377, 1517775403, 6000, 1517404296, 1517404298, 0, 360026, wednesday, 17, TIMEOUT
993953, 589166, other, ulctidrc, 1, [c0911], 1, 1, 2147485696, 52, shared, 1518779950, 1519139959, 6000, 1518688762, 1518688763, 0, 360009, friday, 12, TIMEOUT
1023323, 457496, other, ulcmmrta, 1, [c0630], 1, 1, 2147486720, 52, shared, 1519303539, 1519310101, 900, 1519303400, 1519303400, 0, 6562, thursday, 13, FAILED
869375, 507832, other, ulctidrc, 1, [c1108], 1, 1, 2147485696, 52, shared, 1517425929, 1517785940, 6000, 1517404296, 1517404298, 0, 360011, wednesday, 20, TIMEOUT
895774, 910424, other, ulctidrc, 1, [c1107], 1, 1, 2147485696, 52, shared, 1517775646, 1518135663, 6000, 1517404296, 1517404298, 0, 360017, sunday, 21, TIMEOUT
```

2. Obtener aquellos registros que correspondan a trabajos cuya ejecución haya utilizado un número de CPU comprendido entre 32 y 64 (ambos inclusive).

INSTRUCCIÓN

```
java -jar lazuli.jar -q data.csv MIN_CPUS_ALLOC=32 MAX_CPUS_ALLOC=64
```

SALIDA

```
id_job, priority, account, user, nodes_alloc, nodelist, cpus_req, cpus_alloc, mem_req,
id_qos, partition, time_start, time_end, timelimit, time_submit, time_eligible,
time_suspended, time_elapsed, start_week_day, start_hour, state
1021779, 402102, other, ulccemcm, 2, [c7347-c7348], 48, 48, 122880, 24, thinnodes,
1519568598, 1519928611, 6000, 1519249401, 1519249401, 0, 360013, sunday, 15, TIMEOUT
1023624, 826496, csic, cstsmasc, 2, [c6910-c7223], 48, 48, 122880, 24, thinnodes,
1519425781, 1519445866, 2640, 1519308653, 1519308653, 0, 20085, friday, 23, COMPLETED
1013087, 495435, cesga, prey, 2, [c7235-c7236], 48, 48, 122880, 24, software,
1519131074, 1519131386, 5, 1519123665, 1519123665, 0, 312, tuesday, 13, TIMEOUT
944403, 2506635, other, gfnloper, 2, [c7125-c7126], 48, 48, 122880, 24, reservas,
1518281400, 1518300722, 480, 1518281101, 1518281400, 0, 19322, saturday, 17, COMPLETED
1013349, 494969, cesga, prey, 2, [c7235-c7236], 48, 48, 122880, 24, software,
1519133595, 1519133918, 5, 1519132172, 1519132172, 0, 323, tuesday, 14, TIMEOUT
1021736, 1427016, other, ulcesbfr, 2, [c7103-c7148], 48, 48, 122880, 24, cola-corta,
1519266894, 1519270201, 55, 1519246724, 1519246724, 0, 3307, thursday, 3, TIMEOUT
944097, 535553, eisti, eisti005, 2, [c7226-c7230], 2, 48, 122880, 24, thinnodes,
1518278144, 1518278159, 10, 1518278056, 1518278056, 0, 15, saturday, 16, COMPLETED
```

3. Obtener aquellos registros que correspondan a trabajos cuya ejecución se haya desplegado como mínimo en 8 nodos y que hayan empezado en Lunes o Domingo.

INSTRUCCIÓN

```
java -jar lazuli.jar -q data.csv MIN_NODES_ALLOC=8
START_WEEK_DAY=[monday,sunday]
```

SALIDA

```
id_job, priority, account, user, nodes_alloc, nodelist, cpus_req, cpus_alloc, mem_req,
id_qos, partition, time_start, time_end, timelimit, time_submit, time_eligible,
time_suspended, time_elapsed, start_week_day, start_hour, state
1032043, 1819968, csic, csuntmhg, 8, [c7041-c7042-c7043-c7044-c7045-c7046-c7047-c7048],
128, 192, 122880, 24, cola-corta, 1519657225, 1519663834, 420, 1519598075, 1519598075,
0, 6609, monday, 16, COMPLETED
957409, 1499900, imd03, imd03526, 12, [c6910-c6912-c6915-c6918-c6919-c7302-c7304-c7306-
c7307-c7331-c7336-c7345], 288, 288, 122880, 37, thinnodes, 1518416390, 1518437838, 2400,
1518400745, 1518400745, 0, 21448, monday, 7, FAILED
1032095, 1819968, csic, csuntmhg, 8, [c7001-c7002-c7003-c7004-c7005-c7006-c7007-c7008],
128, 192, 122880, 24, cola-corta, 1519661146, 1519661183, 420, 1519598076, 1519598076,
0, 37, monday, 17, COMPLETED
```

2.4. Resultados

ANÁLISIS DE PREDICCIONES

Se realiza el análisis considerando predicciones realizadas entre el 26 de Abril de 2018 y el 11 de Mayo de 2018.

Para analizar las predicciones se ha utilizado un sistema de scripts que compara la clasificación realizada contra los datos extraídos del CPD tras la terminación de los trabajos. A continuación se presenta dicho sistema de scripts (Anexo 8a):

check.sh

```
#!/bin/bash

pyexport='export_metrics.py'
pycheck='check_results.py'
predfile='predictions_FROM2018_04_26_TO2018_05_11.csv'
jobsfile='jobs_list'
queryfile='output.csv'

cat $predfile | awk 'BEGIN {FS=","} {print $1}' | tail -n +2 > $jobsfile

python $pyexport -out $queryfile -extra-queries $jobsfile
python $pycheck $predfile $queryfile
```

El script en *bash* debe configurarse utilizando las variables que figuran en la primera parte del mismo, el criterio a seguir es el siguiente:

- *pyexport* : Ruta que apunta al script de extracción de métricas.
- *pycheck* : Ruta que apunta al segundo script del análisis, encargado de comparar los datos extraídos con las predicciones realizadas.
- *predfile* : Ruta al archivo de predicciones.
- *jobsfile* : Ruta donde se exportará una lista de manera que en cada línea del archivo figure el id de un trabajo. Estos id corresponden con los que figuran en el archivo de predicciones.
- *queryfile* : Ruta donde se exportarán las extracciones realizadas de la fuente de datos.

El script en *python* que realiza la segunda parte del análisis (la comparación entre las predicciones y los resultados extraídos de la fuente de datos), puede consultarse en el anexo 8a, siendo el nombre del archivo: *check_results.py*

El resultado del análisis del conjunto de predicciones es el que sigue:

```
Success predictions: 2594
Failed predictions: 989
Uncovered cases: 0
Expected accuracy: 0.7703412381708241
Real accuracy: 0.7239743231928552

COMPLETED:
      Success predictions: 1252
      Fail predictions: 690
      But FAILED: 120
      But TIMEOUT: 536
      Accuracy: 0.6446961894953656

FAILED:
      Success predictions: 128
      Fail predictions: 228
      But COMPLETED: 121
      But TIMEOUT: 98
      Accuracy: 0.3595505617977528

TIMEOUT:
      Success predictions: 1214
      Fail predictions: 20
      But COMPLETED: 4
      But FAILED: 0
      Accuracy: 0.9837925445705025
```

Donde los campos generales significan lo siguiente:

- *Success predictions* : Número de predicciones acertadas sobre el total de realizadas.
- *Failed predictions* : Número de predicciones erradas sobre el total de realizadas.
- *Uncovered cases* : Predicciones sobre trabajos cuyo resultado no se ha podido extraer (debería ser 0).
- *Expected accuracy* : Promedio de la precisión de todas las predicciones realizadas.
- *Real accuracy* : Tasa de acierto de las predicciones realizadas.

A mayores, por cada categoría de interés (*COMPLETED*, *FAILED* y *TIMEOUT*) se presenta un desglose donde los campos tienen los siguientes significados:

- *Success predictions* : Número de predicciones acertadas dentro del subconjunto de clasificaciones que corresponden a la categoría. Por ejemplo, en el caso de *COMPLETED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *COMPLETED* acertadamente.
- *Fail predictions* : Número de predicciones erradas dentro del subconjunto de clasificaciones que corresponden a la categoría. Por ejemplo, en el caso de *COMPLETED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *COMPLETED* erradamente.

- *But COMPLETED* : Número de predicciones confundidas con *COMPLETED*. Por ejemplo, en el caso de *FAILED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *FAILED* pero su terminación corresponde con *COMPLETED*.
- *But FAILED* : Número de predicciones confundidas con *FAILED*. Por ejemplo, en el caso de *COMPLETED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *COMPLETED* pero su terminación corresponde con *FAILED*.
- *But TIMEOUT* : Número de predicciones confundidas con *TIMEOUT*. Por ejemplo, en el caso de *COMPLETED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *COMPLETED* pero su terminación corresponde con *TIMEOUT*.
- *Accuracy* : Indica la tasa de acierto de las clasificaciones realizadas para la categoría correspondiente. Por ejemplo, en el caso de *COMPLETED*, hace referencia a la tasa de acierto de aquellos trabajos cuya terminación fue predicha como *COMPLETED*.

Cabe destacar que los datos de las predicciones, a diferencia de lo que ocurre con las notificaciones, incluyen el conjunto de clasificaciones realizadas en bruto. Por tanto, pueden existir varias predicciones sobre el mismo trabajo, que se realizan a lo largo del tiempo para aquellos casos en los que se haya incluido el identificador del trabajo en la *pool* de notificaciones, lo que se debe a la problemática ya comentada con el atributo *time_elapsed*, que requiere que ciertas notificaciones actualicen la clasificación realizada.

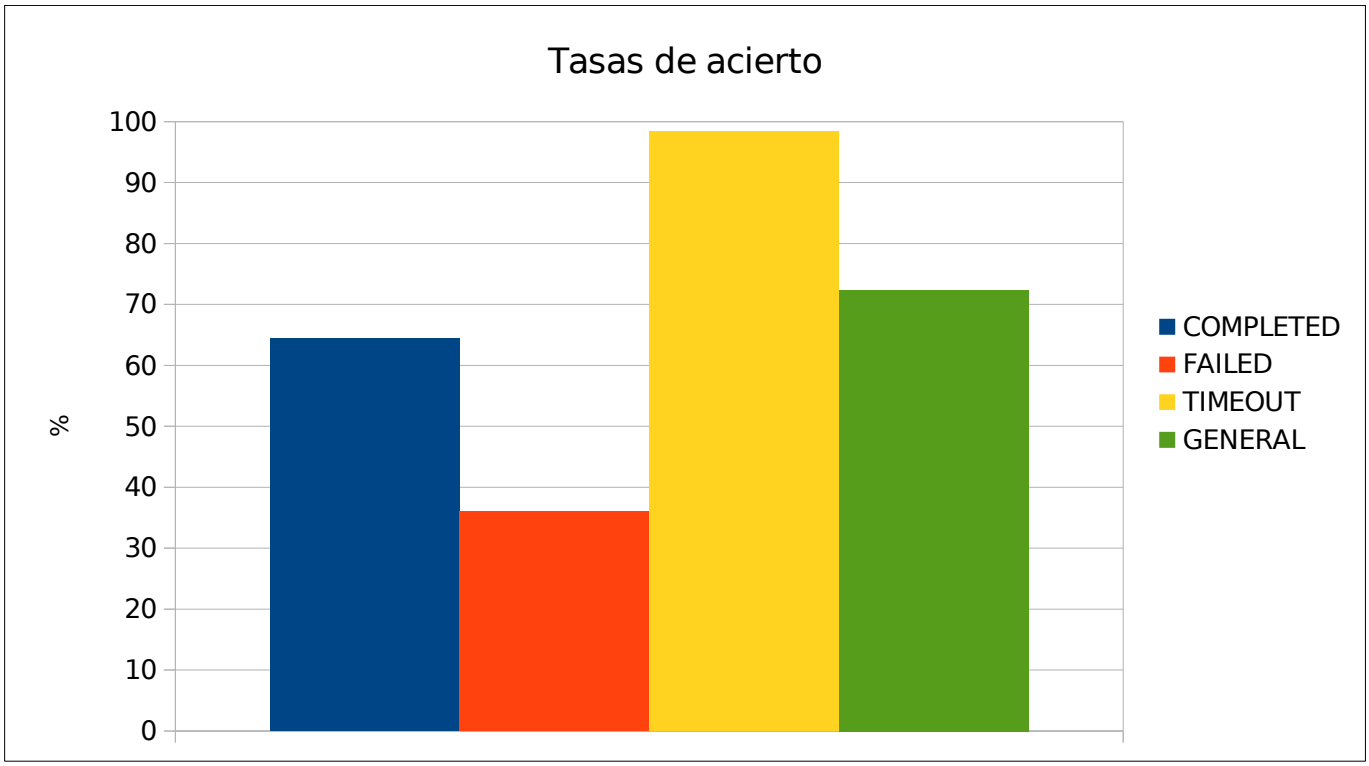
Sin embargo, resulta de interés estudiar el caso de las predicciones en bruto por separado, ya que arroja luz sobre la validez de las predicciones en general, al margen de la evolución temporal y considerando todos los datos y no sólo aquellos que han sido elegidos por su interés para el sistema de notificaciones (que no es más que un subconjunto del conjunto compuesto por el total de las predicciones).

También es importante tener en cuenta que las predicciones han sido realizadas utilizando distintos modelos, pues se han realizado varios lanzamientos de la IA para obtener resultados lo más independientes posible de la eficacia de un modelo puntual. Si bien lo ideal habría sido utilizar intervalos de tiempo más largos de manera que los modelos se fuesen sustituyendo según lo especificado en la configuración, las limitaciones de tiempo han requerido que se trabaje de esta otra manera.

Por último, es menester mencionar que no se han tomado todas las predicciones realizadas puesto que, para evitar saturar el almacenamiento persistente del servidor al cual se ha permitido el acceso para realizar este TFG, se han configurado purgas periódicas de manera que no todos los archivos de predicciones han sido rescatados. No obstante, el tamaño de la muestra es de 3583 registros espaciados en el intervalo temporal especificado al comienzo de este apartado, lo que se considera un volumen de datos lo suficientemente significativo para el análisis realizado.

Para facilitar el estudio de los datos se presentan los siguientes gráficos, cuyo proceso de elaboración puede ser consultado en el archivo *visualizacion.ods* dentro del anexo 8a.

En primer lugar, el gráfico que muestra las tasas de acierto (precisión) en general y según la categoría de clasificación:



De este gráfico se puede deducir fácilmente que cuando se clasifica un registro como *FAILED* la precisión es pésima, mientras que cuando se clasifica como *TIMEOUT* la probabilidad de acierto es muy elevada. Por otra parte, cuando se clasifica como *COMPLETED*, aunque la tasa de acierto supera a la de error, existe una cantidad de fallos significativa. Además, en general, la precisión de las clasificaciones supera el 70%, lo cual aunque mejorable, puede considerarse un resultado aceptable.

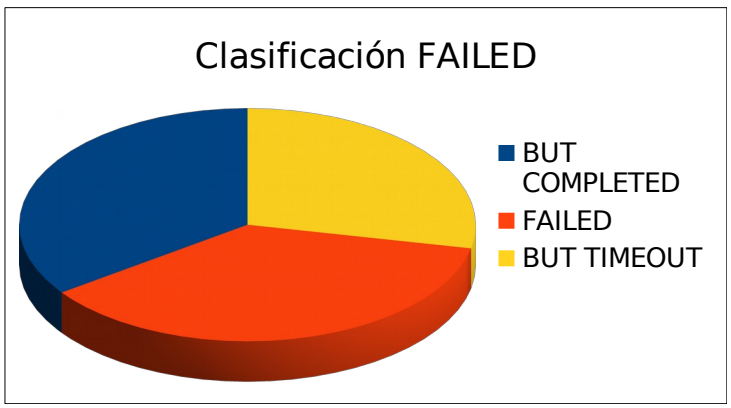
Para entender mejor lo que está ocurriendo se procede a observar en detalle la distribución de confusión y acierto por cada clasificación.

Para *COMPLETED*:

Puede observarse que una cantidad importante de casos clasificados como *COMPLETED* corresponden en realidad a *TIMEOUT*, lo cual podría estar ocurriendo debido a la problemática con *time_elapsed,* que bien puede estar causando que hasta que no transcurra una cierta cantidad de tiempo se espere que un trabajo se complete con éxito pero, una vez superado el umbral de tiempo, se clasifique correctamente como *TIMEOUT*.

Cuando se analicen las notificaciones podrá observarse con más detalle este fenómeno, pues sólo se considera la última predicción de interés y se descartan las anteriores.

Para *FAILED*:



La predicción de terminaciones *FAILED* parece ser la más mutable y la que tiene la mayor tasa de error. De nuevo, es importante esperar al análisis de notificaciones antes de obtener conclusiones precipitadas puesto que, como ya se ha comentado, las predicciones de interés son rastreadas hasta la emisión de un veredicto definitivo o hasta que se pierda el interés según el criterio definido.

Para *TIMEOUT*:

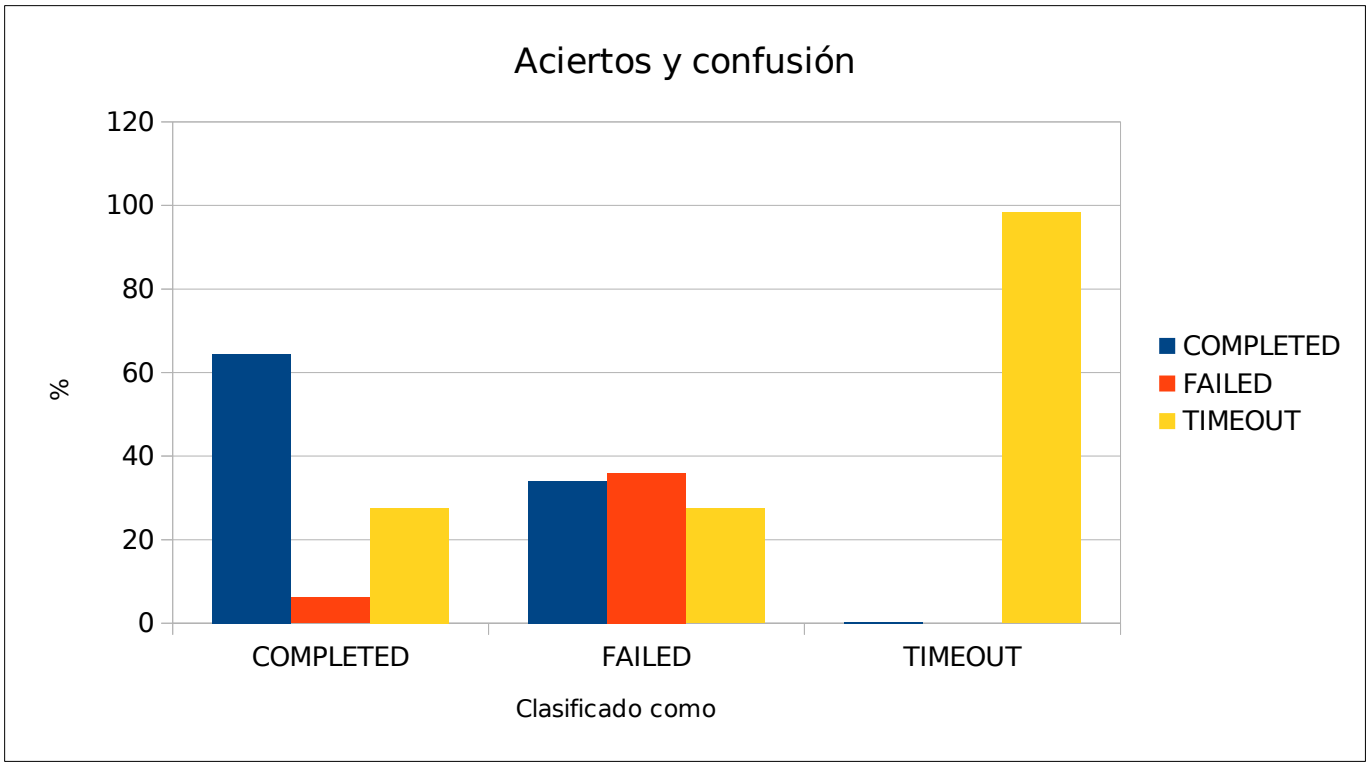
En cuanto a los registros clasificados como *TIMEOUT*, parecen ser los más fiables. Una vez emitido un veredicto de *TIMEOUT* para un trabajo, existe una fuerte tendencia al acierto. Por tanto, a falta del análisis de notificaciones, la detección de *timeouts* parece ser la mayor virtud de la IA.

Finalmente, se presentan el acierto y la confusión de las distintas clasificaciones en un último gráfico que facilita la comparación entre todas las categorías:

(En el primer cluster "COMPLETED", la barra azul corresponde a los casos correctamente clasificados como COMPLETED, la barra roja corresponde a los casos clasificados como COMPLETED que en realidad eran FAILED y la barra amarilla corresponde a los casos clasificados como COMPLETED que en realidad eran TIMEOUT.

En el segundo cluster "FAILED", la barra azul corresponde a los casos clasificados como FAILED que en realidad eran COMPLETED, la barra roja corresponde a los casos correctamente clasificados como FAILED y la barra amarilla corresponde a los casos clasificados como FAILED que en realidad eran TIMEOUT.

En el tercer cluster "TIMEOUT", la barra azul corresponde a los casos clasificados como TIMEOUT que en realidad eran COMPLETED, la barra roja es inexistente pues no se clasificó como TIMEOUT ningún caso FAILED y la barra amarilla corresponde a los casos correctamente clasificados como TIMEOUT.)



Puede observarse a simple vista lo ya comentado; en este primer análisis parece ser que *TIMEOUT* es la categoría que se clasifica con mayor acierto y de una manera más definitiva, *COMPLETED* tiene un funcionamiento simplemente adecuado y *FAILED* es una clasificación de dudosa fiabilidad. No obstante, también se observa que en todas las categorías el número de aciertos supera al número de fallos por categoría, no así al número de fallos total, puesto que en el cluster "*FAILED*" el total de errores supera al total de aciertos aunque ninguna categoría tenga una cantidad de errores que individualmente supere a la de aciertos.

ANÁLISIS DE NOTIFICACIONES

Se realiza el análisis considerando las notificaciones generadas entre el 26 de Abril de 2018 y el 11 de Mayo de 2018.

Para analizar las notificaciones se ha utilizado un sistema de scripts que compara la clasificación realizada con los datos extraídos del CPD tras la terminación de los trabajos. A continuación se presenta dicho sistema de scripts (Anexo 8b):

check.sh

```
#!/bin/bash

pyexport='export_metrics.py'
pycheck='check_results.py'
notfile='notifications_FROM2018_04_26_TO2018_05_11.csv'
jobsfile='jobs_list'
queryfile='output.csv'

cat $notfile | awk 'BEGIN {FS=","} {print $1}' | tail -n +2 > $jobsfile

python $pyexport -out $queryfile -extra-queries $jobsfile
python $pycheck $notfile $queryfile
```

El script en *bash* debe configurarse utilizando las variables que figuran en la primera parte del mismo, el criterio a seguir es el siguiente:

- *pyexport* : Ruta que apunta al script de extracción de métricas.
- *pycheck* : Ruta que apunta al segundo script del análisis, encargado de comparar los datos extraídos con las notificaciones.
- *notfile* : Ruta al archivo de notificaciones.
- *jobsfile* : Ruta donde se exportará una lista de manera que en cada línea del archivo figure el id de un trabajo. Estos id corresponden con los que figuran en el archivo de predicciones.
- queryfile : Ruta donde se exportarán las extracciones realizadas de la fuente de datos.

El script en *python* que realiza la segunda parte del análisis (la comparación entre las notificaciones y los resultados extraídos de la fuente de datos), puede consultarse en el anexo 8b, siendo el nombre del archivo: *check_results.py*

El resultado del análisis del conjunto de notificaciones es el que sigue:

```
Success predictions: 386
Failed predictions: 455
Uncovered cases: 0
Expected accuracy: 0.6881959687405865
Real accuracy: 0.4589774078478002

FAILED:
      Success predictions: 173
      Fail predictions: 444
      But COMPLETED: 265
      But TIMEOUT: 155
      Accuracy: 0.28038897893030795

TIMEOUT:
      Success predictions: 213
      Fail predictions: 11
      But COMPLETED: 3
      But FAILED: 2
      Accuracy: 0.9508928571428571
```

Donde los campos generales significan lo siguiente:

- *Success predictions* : Número de notificaciones acertadas sobre el total de realizadas.
- *Failed predictions* : Número de notificaciones erradas sobre el total de realizadas.
- *Uncovered cases* : Notificaciones sobre trabajos cuyo resultado no se ha podido extraer (debería ser 0).
- *Expected accuracy* : Promedio de la precisión de todas las notificaciones realizadas.
- *Real accuracy* : Tasa de acierto de las notificaciones realizadas.

A mayores, por cada categoría de interés (*FAILED* y *TIMEOUT*) se presenta un desglose donde los campos tienen los siguientes significados:

- *Success predictions* : Número de notificaciones acertadas dentro del subconjunto de clasificaciones que corresponden a la categoría. Por ejemplo, en el caso de *FAILED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como FAILED acertadamente.
- *Fail predictions* : Número de notificaciones erradas dentro del subconjunto de clasificaciones que corresponden a la categoría. Por ejemplo, en el caso de *FAILED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como FAILED erradamente.
- *But COMPLETED* : Número de notificaciones confundidas con *COMPLETED*. Por ejemplo, en el caso de *FAILED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *FAILED* pero su terminación corresponde con *COMPLETED*.
- *But FAILED* : Número de notificaciones confundidas con *FAILED*. Por ejemplo, en el caso de *TIMEOUT*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como *TIMEOUT* pero su terminación corresponde con *FAILED*.

- *But TIMEOUT* : Número de notificaciones confundidas con *TIMEOUT*. Por ejemplo, en el caso de *FAILED*, hace referencia a aquellos trabajos cuyo estado terminal se ha predicho como FAILED pero su terminación corresponde con *TIMEOUT*.
- *Accuracy* : Indica la tasa de acierto de las clasificaciones realizadas para la categoría correspondiente. Por ejemplo, en el caso de *FAILED*, hace referencia a la tasa de acierto de aquellos trabajos cuya terminación fue predicha como *FAILED*.

Cabe destacar que los datos de las notificaciones son un subconjunto, respecto del total de las predicciones, que se genera tomando aquellos registros que cumplen un criterio especificado mediante el archivo de configuración de notificaciones. En este caso concreto, se han utilizado varias ejecuciones con configuraciones sujetas a pequeñas variaciones en lo que al criterio de precisión (*accuracy*) se refiere. No obstante, existe un criterio común y es que sólo se han tenido en cuenta aquellas predicciones que clasificaban un trabajo como *FAILED* o como *TIMEOUT*, pues se ha considerado de interés centrarse en los fallos y dejar a un lado las predicciones asociadas a *COMPLETED*.

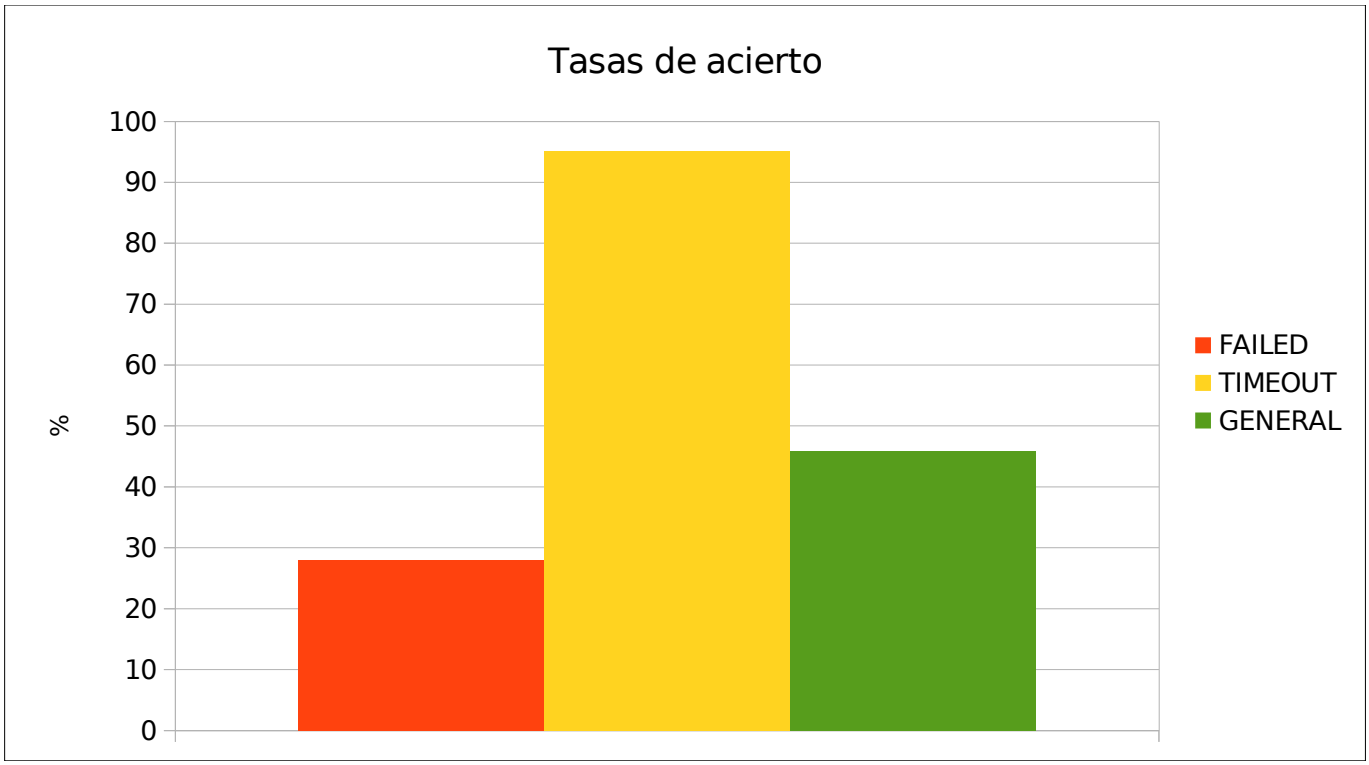
Al centrar las notificaciones alrededor de clasificaciones que corresponden a terminaciones definidas pero no exitosas, se está comprobando la eficacia real del producto para detectar errores. De esta manera, se espera ahondar en la problemática con las clasificaciones *FAILED* que parecía no ocurrir con las *TIMEOUT* y discernir así la utilidad real más precisa del software.

Al igual que ocurría con el análisis de las predicciones, siendo además las notificaciones un subconjunto derivado de estas, es importante tener en cuenta que se han realizado varios lanzamientos de la IA y que, por tanto, se han utilizado modelos distintos para generar las clasificaciones. Por consiguiente, se espera obtener resultados lo más independientes posible de la eficacia de un modelo puntual aunque, puesto que en la generación del modelo se utilizan los datos de los últimos 30 días, seguirá habiendo un subconjunto de los datos de entrenamiento que sea común a los distintos modelos generados.

Cabe destacar que, a diferencia de lo que ocurría con las predicciones, donde para evitar saturar el espacio de almacenamiento persistente del servidor se realizaban borrados automatizados sobre el total de los datos, en el caso de las notificaciones se han conservado absolutamente todas, ya que el volumen de los registros que cumplen con los criterios especificados es mucho más reducido que el total de predicciones. En concreto, se han considerado un total de 841 notificaciones, espaciadas en el tiempo acorde al rango de fechas indicado al comienzo de este apartado.

Para facilitar el estudio de los datos se presentan los siguientes gráficos, cuyo proceso de elaboración puede ser consultado en el archivo *visualizacion.ods* dentro del anexo 8b.

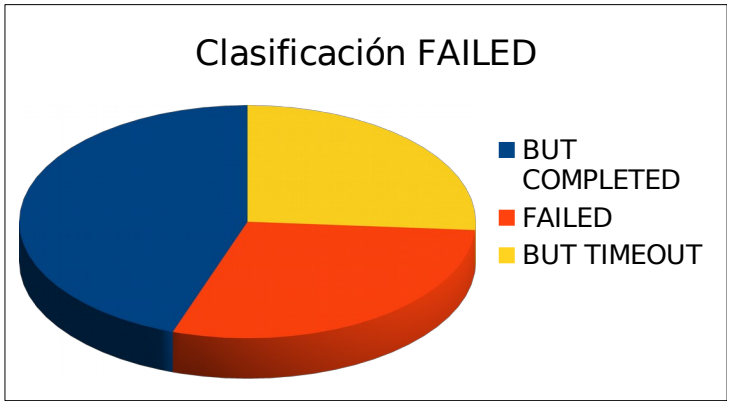
En primer lugar, el gráfico que muestra las tasas de acierto (precisión) en general y según la categoría de la clasificación:



En este gráfico puede observarse que se mantiene la eficacia de las distintas clasificaciones respecto del caso anterior donde se consideraba el conjunto total de las predicciones en lugar de sólo las de interés. En este caso, como ocurría en el anterior, la clasificación de trabajos con estado terminal *FAILED* ni siquiera alcanza el 30% de tasa de acierto, por lo que dicha clasificación parece ser, como ya se había comentado, de una calidad pésima. Por otro lado, la eficacia de la clasificación TIMEOUT, parece ratificarse.

A continuación se procede a observar en detalle la distribución de confusión y acierto para las dos clasificaciones que corresponden a terminaciones no exitosas:

Para *FAILED*:



Las notificaciones de terminaciones *FAILED* son excesivamente imprecisas y constituyen el principal punto débil de este trabajo de fin de grado. Como puede observarse en este diagrama de sectores, la mayor parte de las notificaciones clasificadas como *FAILED* corresponden a falsos positivos donde finalmente la terminación fue exitosa. A su vez, otra buena parte se confunde con errores donde la ejecución termina fuera de tiempo, por lo que en el futuro se recomienda ahondar en esta problemática, tal vez desglosando los fallos en otras categorías (igual que se ha hecho con los *TIMEOUT*, que no se consideran fallos genéricos). No obstante, esto requeriría la modificación de la infraestructura lógica que tendría que pasar a considerar distintos tipos de fallo, lo que escapa al alcance de este proyecto.

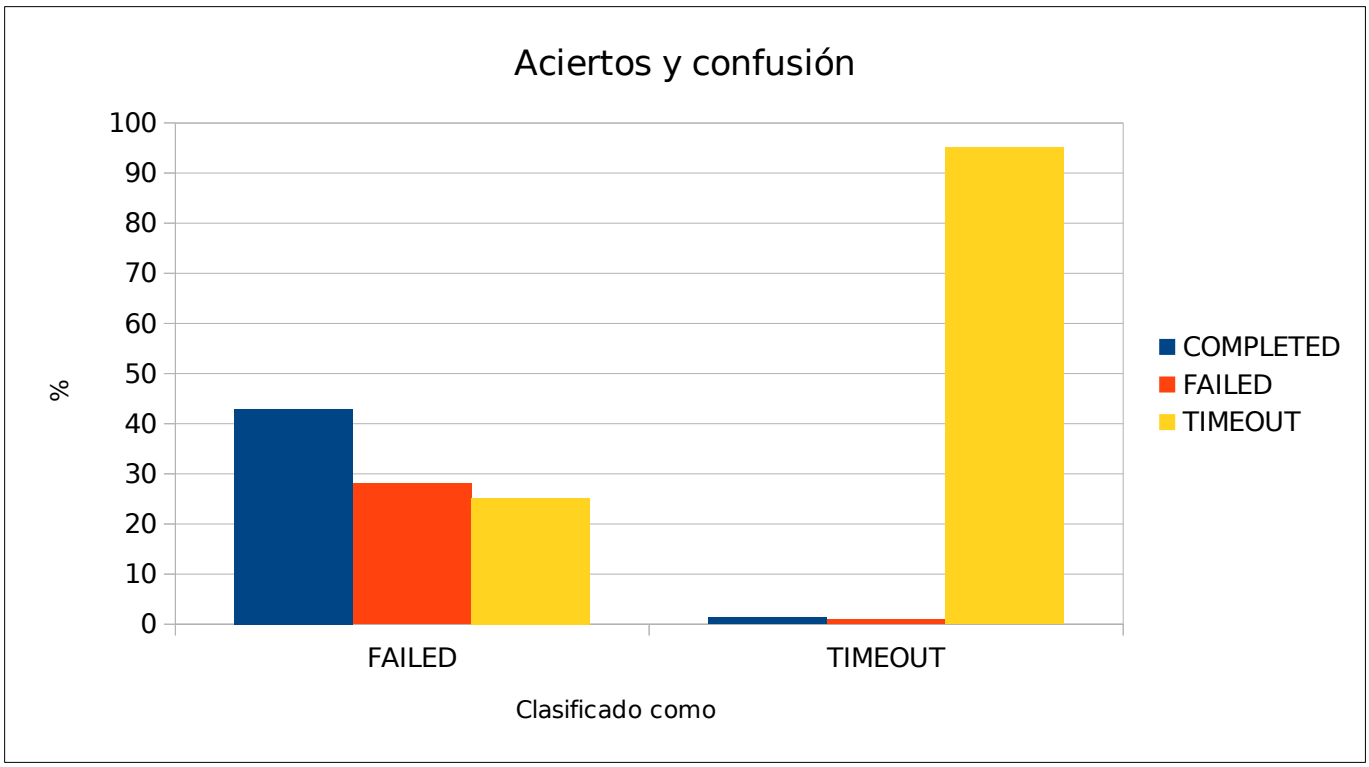
Para *TIMEOUT*:

Por otra parte, las notificaciones de terminaciones *TIMEOUT* tienen una precisión excelente que ronda el 95%, por lo que no es disparatado considerar que este es el punto fuerte del software, la predicción de *timeouts*.

Finalmente, se presentan el acierto y la confusión de las distintas notificaciones en un último gráfico que facilita la comparación entre las categorías *FAILED* y *TIMEOUT:*

(En el primer cluster "FAILED", la barra azul corresponde a los casos clasificados como FAILED que en realidad eran COMPLETED, la barra roja corresponde a los casos correctamente clasificados como FAILED y la barra amarilla corresponde a los casos clasificados como FAILED que en realidad eran TIMEOUT.

En el segundo cluster "TIMEOUT", la barra azul corresponde a los casos clasificados como TIMEOUT que en realidad eran COMPLETED, la barra roja corresponde a los casos clasificados como TIMEOUT que en realidad eran FAILED y la barra amarilla corresponde a los casos correctamente clasificados como TIMEOUT.)



De nuevo se observa lo ya expuesto, la gran diferencia entre las clasificaciones *FAILED* y *TIMEOUT*, teniendo la primera una precisión pésima y gozando la segunda de una tasa de acierto excelente.

Así pues, en el análisis de notificaciones quedan ratificados los resultados obtenidos durante el análisis de predicciones.

3. Conclusiones

3.1. Conclusiones principales

LA NECESIDAD DE DOMINAR DIFERENTES HERRAMIENTAS

En el desarrollo del trabajo ha quedado patente la necesidad de dominar diferentes herramientas. En la vida de un ingeniero informático conocer distintas técnicas y manejar distintos lenguajes de programación/scripting es una cualidad prácticamente imprescindible para desarrollar una gran cantidad de proyectos que requieren de varios componentes capaces de asumir diversas responsabilidades.

Sólo para el desarrollo de este trabajo de fin de grado, que no es más que una semilla -o los primeros pasos- de lo que sería un proyecto real, ha sido necesario utilizar cinco lenguajes distintos: *bash*, *python*, *SQL*, *R* y *Java*.

Así mismo, como el trabajo se desarrolla tanto en local como contra la infraestructura de un CPD, también ha sido necesario el uso de distintas herramientas y tecnologías, desde distintos IDEs y editores de texto como: *Libre Office*, *Eclipse*, *VIM*, *Geany y Mousepad*; hasta utilidades para el trabajo remoto y la ejecución desde línea de comandos como: *xfce4-terminal*, *ssh, scp*, *anaconda*, *sqoop* y *SLURM*.

También es necesario mencionar el análisis de algoritmos y la necesidad de conocer las distintas técnicas para desarrollar software capaz de resolver problemas y complicaciones derivadas de estos. En este caso concreto se tiene por un lado el uso de la algoritmia *C5.0*, cuya documentación ha habido que consultar para entender como funcionaba y poder programar en *Java* el parseo y aplicación de los archivos de reglas generados. Por el otro, la necesidad de realizar gestión de la concurrencia utilizando el mecanismo de sincronización ofrecido por *Java* para emular las técnicas de *mutex* y *semáforo*. A su vez, programar el parseo y la escritura de archivos también es una cualidad imprescindible para cualquier programador, ya que suele ser un requisito de la gran mayoría de productos de software el poder acceder al almacenamiento persistente de una computadora para realizar operaciones de lectura y escritura.

A mayores, estadísticos como *Kappa* o conceptos como la *matriz de confusión o tasas de acierto y error* son un claro ejemplo de que el mundo de la computación está íntimamente ligado a las matemáticas y el dominio de ciertos conceptos de este campo es imprescindible para un ingeniero informático.

LA IMPORTANCIA DE LA INTEGRACIÓN DE DISTINTOS COMPONENTES Y EL PRODUCTO FINAL

La capacidad de comprender y utilizar distintas herramientas, como se ha comentado en el punto anterior, es imprescindible para cualquier ingeniero informático. No obstante, no basta simplemente con saber utilizar distintas técnicas y utilidades por separado, sino que es necesario integrarlas en un mismo producto que pueda ser utilizado por el usuario de manera que -aun estando compuesto por varias partes diferenciadas- pueda ser utilizado como un todo o un único producto.

Para tal fin, observar y analizar otras herramientas es un buen punto de partida para entender como debería ser la interacción con el usuario final. En el caso de este TFG, el producto ofrecido se centraliza en una aplicación de línea de comandos para ser ejecutada en entornos *Linux* como un programa en *Java*, cuyo punto de entrada a la ejecución es un archivo *jar.* Por tanto, se imita el modelo de parámetros propio de las aplicaciones para sistemas Linux, donde tras la invocación del ejecutable se añaden argumentos precedidos por un guión que concretan la ejecución a realizar. A su vez, también se ha recurrido al uso de archivos de configuración persistentes para definir los parámetros que permiten configurar el servicio de recolección y clasificación de datos. De esta manera, se respeta cierto estándar y se facilita el trabajo con el software.

Es posible resumir este apartado en dos conclusiones sencillas:

- Las distintas partes deben comportarse como un todo ante el usuario final. Pues esto facilita el uso, aprendizaje y comprensión del producto generado.
- El todo debe basarse -aunque pueden tomarse ciertas libertades, por supuesto- en aplicaciones ya existentes para facilitar al usuario el que se acostumbre a la herramienta.

DIFICULTADES DEL ENTORNO REAL FRENTE AL ENTORNO ACADÉMICO

Por encima de cualquier otra conclusión, la experiencia más destacable de este TFG ha sido -sin lugar a dudas- la diferencia entre trabajar en un entorno académico y un entorno real.

En el contexto académico suelen utilizarse sistemas simulados, conjuntos de datos adecuados para ilustrar los problemas a tratar, técnicas concretas para situaciones concretas y, en general, todo está preparado para un correcto desempeño del alumno.

Sin embargo, en un entorno real, una buena parte del tiempo del desarrollo no se dedica al estudio de los algoritmos, ni a la programación del software, tampoco a la realización de los distintos experimentos ni a la evaluación directa de resultados; sino que se dedica a resolver incidencias y situaciones anómalas o inesperadas que van surgiendo sobre la marcha.

Por ejemplo, al recolectar datos en el modo de funcionamiento para la evaluación final del software, los datos que registra el sistema no corresponden a situaciones como las de entrenamiento o prueba donde los datos eran estáticos. Algunos usuarios pueden encolar varias veces el mismo trabajo en una época concreta, dando lugar a situaciones inesperadas como la repetición de registros, lo que introduce valores redundantes que desvirtúan la evaluación de los resultados.

Por otro lado, también han surgido inconvenientes que han requerido adaptar el sistema mediante el uso de variables de entorno para poder ejecutar el software en el CPD, mientras que al trabajar en local estos problemas ni siquiera eran tenidos en cuenta.

A su vez, la instalación de paquetes en local es relativamente sencilla, mientras que en el entorno real se requiere de una mayor cautela para respetar la configuración actual de la infraestructura. Así mismo, los gestores de paquetes y posibilidades de instalación están mucho más limitados. En el caso concreto de este TFG ha sido necesario generar un espacio virtual dentro del *home* (directorio personal del usuario en entornos *Linux*) para poder instalar paquetes locales usando *anaconda* y el software no puede ser ejecutado fuera de ese contexto virtual, que debe ser cargado tras la conexión *ssh*; de lo contrario, no se podría utilizar el script en *R* al depender de paquetes que no están disponibles para el sistema global.

En resumen, la experiencia de trabajar en un entorno real es bastante más complicada y aporta dificultades extra respecto del entorno académico.

3.2. Crítica sobre el trabajo

OBJETIVOS LOGRADOS

El objetivo de construir un modelo capaz de predecir el estado final de la ejecución de un trabajo en base a los atributos que lo definen por parte del sistema gestor de colas, puede considerarse parcialmente cumplido.

No todas las clasificaciones realizadas son siempre acertadas y, como se ha podido observar en base a los modelos analizados en este documento, existen sistemas de clasificación que funcionan con bastante precisión y otros que suelen tener algún punto débil y tienden a cometer errores en ese tipo de predicciones.

Esto no se considera un fracaso, puesto que al estar trabajando con un sistema de producción vivo y en constante mutación, el aprendizaje realizado que se vuelca en un archivo persistente (el archivo de reglas del modelo) pierde su validez cuando se produce algún tipo de cambio sustancial, a saber:

- Usuarios activos pasan a estar inactivos y otros que estaban inactivos pasan a estar activos, encolando distintos trabajos cuya clasificación no es adecuada en base a lo aprendido por el modelo.
- La infraestructura del CPD puede sufrir incidencias que ocasionen fallos mientras no sean resueltas y estos pueden influir tanto en el aprendizaje, dando lugar a un modelo que ya no funcionará con tanta eficacia cuando el sistema vuelva al funcionamiento correcto, como en la aplicación, de manera que un modelo “sano” pierda parte de su eficacia al estar prediciendo en un contexto con incidencias.
- Cambios de proyecto. Si un modelo se ha construido durante una etapa en la que los trabajos encolados corresponden a unos proyectos (aprendizaje, investigación, proyectos industriales, etc.) concretos, cuando los trabajos que se encolen correspondan a otros proyectos, el modelo seguirá utilizando lo aprendido mientras no se actualice y sus predicciones ya no resultarán tan fiables.

Sin embargo, en las condiciones deseadas (entorno no real o de desarrollo) los modelos funcionan y permiten realizar clasificaciones con unas tasas de acierto más que deseables (como puede observarse en el apartado donde se exponen los experimentos, concretamente cuando se usa el modelo basado en el conjunto de datos *training_month2*).

A su vez, es importante mencionar que, como se puede observar en el anexo 4 (donde figuran los datos de los experimentos realizados en el entorno académico), todos los modelos construidos para su análisis teórico han cumplido la mayoría de requisitos formales:

- Todos han presentado un valor de *kappa* por encima del 0.1

- Por el volumen de datos escogido, todos cubren una amplia gama cuando se trata de datos categóricos con muchos valores posibles, como es el caso del atributo *user*.
- La precisión o tasa de acierto más reducida ha sido del 90% (para *training_month2*), por tanto puede considerarse que este criterio ha sido satisfactoriamente cumplido por todos los modelos en el plano teórico.
- En cuanto a la matriz de confusión, se había mencionado que lo ideal sería que para los clasificadores *COMPLETED*, *FAILED* y *TIMEOUT*, hubiese más aciertos que errores. Esto no se ha cumplido en su totalidad ya que tanto con los datos *training_week1*, como con *training_week2* y *training_month1*, ha ocurrido que uno de los clasificadores (y sólo uno) ha tenido más errores que aciertos. No obstante, si se ha cumplido con los modelos generados a partir de los datos *training_week3* y *training_month2*.

Sobre las predicciones realizadas en el entorno real, es importante destacar que, si al comienzo de este apartado se consideraba el objetivo parcialmente cumplido, esto se debe al hecho de que los errores generales (categoría *FAILED*) no han llegado a ser predichos con eficacia con los modelos usados en el contexto de producción (aunque habría que mantener el software funcionando durante varios meses para que considerar los experimentos concluyentes y no se dispone de tanto tiempo para el desarrollo de este TFG). No obstante, la predicción de *timeouts* ha tenido una tasa de aciertos excelente y puede considerarse que en lo tocante a este tipo de errores, el sistema ha sido un éxito en las pruebas realizadas en el entorno real.

Teniendo esto en cuenta, se justifica la premisa inicial de que el software funciona parcialmente, ya que se han llegado a obtener resultados interesantes cuya aplicación podría considerarse de utilidad. Pese a todo, el sistema necesita un análisis más profundo para aplicar las mejoras oportunas y subsanar la inestabilidad y problemática que varían según el modelo, como se puede observar en los experimentos realizados en la etapa de desarrollo. A mayores, se requiere modificar -también- el despliegue en el entorno real para adecuarlo a las situaciones anómalas y cambios constantes a los que está supeditada la actividad de un CPD.

Sobre lo anteriormente expuesto se desarrollara un análisis y unas propuestas de mejora en el apartado de *crítica sobre el trabajo*, así como en la sección de *trabajo futuro*.

Finalmente, el sistema de consultas o filtros, tanto cuando se aplican sobre las notificaciones como cuando se aplican sobre los archivos de datos en bruto, funciona correctamente y puede considerarse absolutamente cumplido.

PROBLEMAS DE PRECISIÓN VARIABLE

Considerando las imprecisiones expuestas en el punto anterior, uno de los puntos débiles del proyecto es la frecuencia del proceso de aprendizaje.

Actualmente el software realiza las predicciones principales con el conjunto de datos extraído del último mes, que se renueva una vez cada 30 días. El problema es que los proyectos vigentes, incidencias y demás situaciones que supongan una alteración respecto del estado de la infraestructura en el momento de aprendizaje, no son algo regular que ocurra una vez al mes ni en intervalos de tiempo conocidos.

Por tanto, una de las mejoras que supondría un considerable aumento de la calidad del modelo consistiría en estudiar las diferentes anomalías que se producen en el CPD y, una vez extraído el conocimiento necesario, ajustar el aprendizaje del modelo de manera que sea más adaptativo y se actualice o renueve cuando ocurran este tipo de situaciones.

No obstante, esto escapa del ámbito que se pretendía cubrir con el TFG y, aunque se reconoce como uno de sus principales puntos débiles, no ha sido corregido puesto que tomaría una gran cantidad de tiempo familiarizarse con la infraestructura del centro y, a mayores, posiblemente sería necesario construir otra capa de software que se comunicase con el desarrollado en este trabajo para aportarle la información que le permita ajustar el proceso de aprendizaje considerando las distintas eventualidades posibles.

PREDICCIONES FAILED EN ENTORNO ACADÉMICO

Es importante analizar el resultado en el entorno de producción sin dejar de lado los resultados en el entorno académico. En el caso del experimento realizado en el CPD se obtuvo un clasificador que da un resultado excelente con las clasificaciones de errores *TIMEOUT* pero que experimenta problemas de precisión cuando se trata de clasificaciones de errores genéricos *FAILED*.

Si se atiende a los experimentos llevados a cabo en el plano teórico, esta realidad podría corresponder a modelos como los generados con los datos de entrenamiento *training_month2* y *training_week3*. No obstante, es menester recordar que con los modelos derivados de los conjuntos de datos *training_month1 y training_week2* se ha dado justo la situación inversa: El sistema de clasificación tiene una mayor precisión cuando se trata de detectar terminaciones *FAILED* que *TIMEOUT*.

Por tanto, cabe esperar que en el entorno de producción (del cual han sido extraídos los datos para usarlos en el ámbito académico) tarde o temprano termine por darse la misma situación y que, según los datos utilizados en el aprendizaje, puedan generarse modelos que terminen por predecir con mucha mayor eficiencia los fallos genéricos *FAILED* que los fallos concretos *TIMEOUT*.

Además de las propuestas que se comentan en los siguientes apartados, no conviene olvidar que, de ser posible, sería muy recomendable deshacerse de la categoría *FAILED* y dividirla en otras categorías que concreten el tipo de error a detectar, en lugar de considerar un error genérico. Se espera que esto permita establecer una correlación más fuerte entre el impacto de los atributos y las categorías en que se clasifican los registros. Sin embargo, para realizar dicha modificación, habría que manipular la infraestructura del CPD para que registre en la fuente de datos los errores considerando más categorías de las que actualmente maneja. Por desgracia, esto último escapa del ámbito del TFG.

PROBLEMÁTICA CON MODELOS FUNDAMENTADOS PRINCIPALMENTE EN TIME_ELAPSED

Yendo a lo concreto, existe un problema con uno de los atributos utilizado por los modelos de clasificación: *time_elapsed*

Este atributo es distinto del resto puesto que mide el tiempo transcurrido desde que comenzó la ejecución del trabajo hasta que o bien termino, o bien se llega al momento en que se recolectó el dato.

Por consiguiente, cuando se construyen modelos que se basan fundamentalmente en el campo *time_elapsed* las predicciones realizadas sobre un trabajo serán más fiables cuanto más avanzado esté este, ya que un trabajo que haya comenzado recientemente tiene un valor de *time_elapsed* más alejado del valor que correspondería a este atributo en el momento de su terminación, que es la manera en que se ha utilizado este campo para entrenar el modelo.

Para solventar esto, se ha aplicado una solución parcial que consiste en mantener un rastreo de las predicciones en curso que se consideran de interés por parte del sistema de notificaciones. De esta manera, mientras una predicción siga resultando de interés y no haya concluido, la notificación correspondiente será actualizada. El principal inconveniente de esta solución es que mientras no se haya avanzado en el tiempo, se realizarán predicciones que en un primer momento no serán tan fiables y, para obtener predicciones más fiables, será necesario esperar.

Podría haberse prescindido del campo *time_elapsed*, pero se ha observado que tiene un elevado grado de significatividad dentro de varios de los modelos generados, por tanto se ha optado por la solución parcial inicialmente propuesta.

De cara a futuras ampliaciones del proyecto, podría considerarse el uso de predicciones sobre series temporales analizando las métricas de rendimiento y funcionamiento de los nodos en que se ejecutan los trabajos. De esta manera, sería necesario un tiempo antes de predecir potenciales errores; sin embargo, este sistema se complementaría con el elaborado en este TFG y, puesto que también requeriría de un tiempo para analizar el comportamiento de los nodos computacionales y predecir posibles anomalías o errores, la solución parcial adoptada terminaría por ser la más adecuada.

3.3. Seguimiento de la planificación

Una de las decisiones tomadas en este proyecto ha sido la de utilizar un sistema de planificación más flexible, basado en los postulados del manifiesto AGILE [19]. Esta decisión se ha tomado porque, al trabajar en un entorno real, se preveía -y efectivamente, así ha sido- que una planificación rígida no podría ser cumplida y tendría que estar siendo constantemente revisada, debido a los imprevistos que han surgido y la necesidad de poder probar distintos enfoques y planteamientos de una manera más flexible.

Es necesario explicar que en esta memoria no se ha incluido cada pequeño cambio de enfoque, ni cada incidencia surgida; especialmente cuando estás han venido derivadas del hecho de tener que adaptarse a un entorno de producción real. Principalmente debido a la limitación de máximo 90 páginas para la memoria, que si bien soportaría que se hubiesen incluido algunas problemáticas, no sería extensión suficiente para comentarlas todas y, en segundo lugar, porque supondría una gran inversión de tiempo y recursos en la documentación de cada detalle, sobre todo teniendo en cuenta que una gran cantidad de estos pertenecen más al ámbito de la administración de sistemas que al del desarrollo de un software de inteligencia artificial (si bien al final son disciplinas que suelen terminar entrelazándose en el mundo de la ingeniería informática).

El orden presentado en el calendario se ha respetado pero ha sido necesario operar varias veces entre distintos hitos, de manera que se han realizado modificaciones y repeticiones fuera de los plazos temporales marcados. A continuación se expone un análisis en detalle de cada elemento de la planificación:

1.1. Programación de scripts para la extracción y transformación de datos

De 12-03-2018 a 25-03-2018

El paso inicial, tras la toma de contacto con las herramientas, ha sido cumplido dentro de plazo. No obstante, los scripts generados han tenido que ser actualizados también durante las iteraciones 3.1 y 3.2 para el desarrollo del software en *Java,* principalmente añadiendo scripts en bash a modo de *wrappers* para la ejecución desde *Java* del script de extracción escrito en *python;* aunque también añadiendo algunas funcionalidades nuevas al script de extracción como los parámetros *-running-only* y *-extra-queries*.

1.2. Construcción de modelos con los datos extraídos.

De 19-03-2018 a 01-04-2018

La construcción de los primeros modelos, es decir, aquellos que han sido utilizados para el análisis de experimentos, fue realizada dentro de plazo. No obstante, durante el análisis los modelos fueron reconstruidos en varias ocasiones. Sin embargo, como dichas reconstrucciones fueron realizadas sobre los mismos datos y dando lugar a los mismos modelos, no se considera que pertenezcan a este hito, sino que se entienden como una mera repetición de la ejecución del script realizada dentro del análisis.

1.3. Elección del modelo/s más adecuados

De 26-03-2018 a 01-04-2018

La primera parte de la decisión sobre los modelos más adecuados se realizó en plazo pero, el análisis posterior de los experimentos incluidos en esta memoria se ha realizado fuera del plazo marcado; puesto que se ha decidido esperar hasta el 1 de Abril para recolectar los datos hasta dicho día.

2.1. Probar el modelo con los datos de prueba

De 02-04-2018 a 08-04-2018

La prueba de los modelos con datos de prueba se ha realizado conjuntamente con la segunda etapa de la elección de los modelos más adecuados (ocurrida en los primeros días de Abril) y corresponde con el análisis de los experimentos expuestos en la memoria.

2.2. Probar el modelo con nuevos datos

De 02-04-2018 a 08-04-2018

La prueba de los modelos con nuevos datos ha sido realizada dentro de plazo, pero ha sido una operación recurrente a lo largo de todo el proyecto.

Realmente debería considerarse como un hito atemporal extendido a lo largo de las distintas iteraciones para validar el valor de los aprendizajes realizados.

3.1. Desarrollo base del software de inteligencia artificial en *Java*.

De 09-04-2018 a 29-04-2018

El desarrollo del software se ha realizado dentro de plazo.

3.2. Integración del modelo predictivo en la IA.

De 23-04-2018 a 06-05-2018

La integración del modelo predictivo se ha realizado dentro de plazo pero en la etapa de despliegue en el entorno de producción ha sido necesario realizar modificaciones para corregir pequeños errores.

4.1. Añadir funciones de consulta al software.

De 30-04-2018 a 13-05-2018

Las funciones de consulta al software comenzaron antes de lo previsto y terminaron antes de que comenzase siquiera el mes de Mayo, por tanto este hito puede considerarse cumplido con antelación.

4.2. Añadir sistema de notificaciones al software.

De 07-05-2018 a 13-05-2018

El sistema de notificaciones quedó implementado a finales del mes de Abril, por lo tanto se ha completado dicho hito antes de plazo. Sin embargo, al desplegar en el entorno real, fue necesario realizar una serie de correcciones para que las predicciones y notificaciones se ejecutasen de manera correcta. Dichas correcciones ocurrieron también dentro de plazo.

5.1. Puesta a punto de la IA y de sus distintas funcionalidades, solventando los errores que puedan surgir en el proceso.

De 07-05-2018 a 20-05-2018

Con el fin de no entorpecer la redacción de la memoria se ha modificado el plazo, aprovechando que el sistema de notificaciones quedó implementado a finales del mes de Abril, que ha pasado a ser:

De 26-04-2018 a 14-05-2018

5.2. Análisis de los resultados obtenidos en el entorno real/de producción

Finalmente, se ha añadido un nuevo elemento a la planificación que no constaba en la original, debido a que se disponía del tiempo necesario para realizarlo.

Este hito corresponde al análisis de los resultados obtenidos a partir de la ejecución en el entorno real, que si bien no han podido ser concluyentes, puesto que habrían sido necesarios como mínimo 2 o 3 meses para analizar el funcionamiento real del software final, han permitido hacerse una idea aproximada de la calidad del mismo.

El análisis de los resultados comienza a mediados de la fase de puesta a punto y concluye poco después, de manera que abarca el siguiente intervalo temporal:

De 07-05-2018 a 17-05-2018

3.4. Trabajo futuro

INTEGRACIÓN CON OTRAS TÉCNICAS DE INTELIGENCIA ARTIFICIAL

Como se ha comentado en el apartado de crítica sobre el trabajo (3.2), existe una imprecisión variable en la aplicación del modelo que se origina al estar trabajando en un entorno real en constante cambio. Para conseguir que la precisión de las predicciones sea más estable y no esté sometida al influjo de las variaciones, se necesita lograr cierta capacidad de adaptación. Sería interesante utilizar otras técnicas de inteligencia artificial, tanto para realizar la clasificación con otro tipo de datos y enfoque, como para extraer otro tipo de información que permita realizar los aprendizajes conforme se vayan detectando cambios significativos en el estado de la infraestructura computacional o, incluso, gestionar el aprendizaje para utilizar distintos modelos según se clasifique el momento y tipo de trabajo a predecir.

Existen dos ideas principales que se consideran para mejorar el proyecto en el futuro:

- *Análisis de los nodos computacionales donde corren los trabajos para detectar anomalías en el tiempo*. De esta manera, cuando se detecten anomalías en ciertos nodos computacionales, será posible incorporar esta información al sistema de clasificación para predecir aquellos casos en los que los errores se produzcan por fallos de hardware o incidencias en la red.

  La detección de anomalías en series temporales es algo que se realiza en distintas empresas actualmente, entre las que se encuentra, por ejemplo, Twitter [17]. Así pues, es una opción a tener en cuenta como mínimo para ser estudiada en el futuro.

- *Uso inteligente de distintos modelos de clasificación*. Como se comentó anteriormente, para refinar el uso de los modelos, podría utilizarse una inteligencia basada en redes neuronales [18] que se entrene para clasificar los distintos encolamientos de manera que se utilice el tipo de modelo más adecuado para cada clasificación. De esta forma, en lugar de tener un sólo modelo principal para efectuar todas las predicciones y realizar un aprendizaje que cambia el modelo cada x intervalo de tiempo (en este caso se ha decidido que sea cada 30 días), se podrían generar varios modelos y utilizar cada uno para el conjunto de trabajos más adecuado, considerando una configuración independiente y optimizada para cada caso.

  No obstante, entrenar esta red neuronal para que sea capaz de tomar decisiones en el entorno real, teniendo en cuenta que aunque haya trabajos cuya ejecución dure horas, otros se demoran más de un día o -incluso- días, requiere de unos tiempos que se escapan de lo disponible para la realización de este TFG.

A mayores, cabe considerar la posibilidad de probar con otros métodos de clasificación como los ya comentados: *Algoritmo de cobertura* y *ADTree* (árbol de decisión alternativo); basados en *ID3* al igual que *C5.0*. Alternativas como *JRip* [13], *Nnge* [14] , *OneR* [15] o *Ridor* [16] podrían ser interesantes también, especialmente si consideramos que se basan en una idea de partida distinta a los primeramente expuestos, todos fundamentados en *ID3* (inducción mediante árboles de decisión).

En el caso de *JRip*, por ejemplo, se trata de una versión mejorada de *IREP*, por lo que se basa en un procedimiento iterativo que divide el conjunto de datos de entrenamiento en dos subconjuntos: crecimiento y poda. En cada iteración, se aplica sobre el subconjunto de datos de crecimiento una operación derivada del subconjunto de poda, que se elige considerando aquella operación que ofrezca la mayor reducción de error sobre el subconjunto de datos de poda, y de la que se espera que sirva para deshacerse de reglas sencillas (por ejemplo, aquellas que contienen una única condición), obteniendo un conjunto final preciso y reducido. Se considera que la aplicación del algoritmo termina cuando no es posible extraer un operador de poda sin que esto de lugar a un incremento del error considerando el conjunto de datos de poda.

Finalmente, como se trata de agrupar en categorías, podría resultar de interés considerar algoritmos de *clustering* propios del mundo del aprendizaje computacional, tales como es el caso de *k-means* que puede utilizarse tanto con la media (*k-means* propiamente dicho), con la mediana (*k-medians*) o con centroides (*k-centroids)* o medoides (*k-medoids*) [22]. Estos algoritmos se basan en calcular la distancia de cada registro respecto a una referencia para agruparlo en alguna de las particiones (puede indicarse el número de particiones deseadas mediante el parámetro *k*, de ahí los nombres). En el caso que nos atañe habría que hacer corresponder cada partición con la categoría oportuna, a saber: *COMPLETED*, *FAILED, TIMEOUT* y *UNKNOWN*.

INTERFAZ GRÁFICA

Uno de los aspectos más característicos del software actual es el uso de interfaces gráficas para simplificar y facilitar la interacción con el usuario. En el caso del software desarrollado tenemos por un lado la gestión del servicio, que realmente no se vería especialmente beneficiada por el uso de una interfaz gráfica de usuario, pero por el otro tenemos tanto el módulo de consultas sobre los datos como la observación de las notificaciones, que en el estado actual del software simplemente se exportan al sistema de archivos en formato CSV.

La principal mejora que se plantea en este aspecto sería la construcción de una interfaz gráfica de usuario, preferiblemente en forma de aplicación web para poder ser utilizada desde orígenes remotos, desde la cual se cubran tres aspectos:

- *Gestión del servicio de clasificación de trabajos encolados*. Esta característica sería la menos relevante, puesto que dicha gestión deberá ser realizada por técnicos que deberían estar familiarizados con el uso de herramientas de línea de comandos. No obstante, la gestión del servicio (arrancar, detener, reiniciar, consultar estado) podría ser realizada perfectamente desde la aplicación web.

- *Visualización de las notificaciones*. Actualmente las notificaciones se exportan al sistema de archivos en formato CSV, sin embargo, en la interfaz gráfica de usuario, podría utilizarse un sistema de visualización basado en tablas con posibles filtros y un sistema de colores o iconos que corresponda con la precisión de cada predicción.

  Por ejemplo: En tono rojo aquellas clasificaciones cuya precisión esté por debajo del 50%, en naranja aquellas que estén entre el 50% y el 70%, en amarillo las que estén entre el 70% y el 80%, en azul las que estén entre el 80% y el 90% y en verde las que estén por encima del 90%. Otra opción sería asociar los colores a los estados en lugar de a la precisión.

También sería interesante dar soporte a operaciones de ordenación por campos en la tabla, para ordenar las notificaciones en base a atributos temporales (orden cronológico de trabajos), en base a la precisión, en base a el orden secuencial de identificadores de trabajo, agrupar por estados, etc.

- *Sistema de consultas*. El producto final que se entrega permite realizar consultas sobre archivos de datos (recolecciones en bruto, predicciones y notificaciones) utilizando la linea de comandos. Sería interesante construir un módulo en la aplicación web que permita realizar estas consultas mediante un asistente gráfico. Resulta más cómodo y, para aquellos interesados que deseen consultar los datos, y no tengan soltura en el manejo de herramientas para terminal, se conseguiría una importante mejora en la accesibilidad del producto.

APLICACIÓN EN OTROS ÁMBITOS

Por la naturaleza del sistema utilizado para realizar las predicciones, el software generado podría someterse a ligeras modificaciones de manera que sería sencillo adaptarlo para obtener resultados en otros ámbitos.

En el caso de este TFG se ha tomado como caso el sistema de encolamientos de trabajos en un centro de computación distribuida, puesto que se consideró de interés disponer de un entorno real en el que desplegar el software para comprobar su funcionamiento en un contexto de producción. No obstante, aquellos casos que puedan ser cubiertos por la algoritmia C5.0, pueden ser cubiertos también por este software. Algunos ejemplo de otros ámbitos donde se espera que se pueda adaptar con eficacia el producto final derivado de este trabajo, son:

- *Diagnóstico médico*. Donde, a partir de unos atributos de interés concretos (podrían corresponder con los marcadores obtenidos a través de una analítica sanguínea), se pretenda conocer si el paciente padece una enfermedad (diagnóstico positivo) o no (diagnóstico negativo).
- *Clasificación de clientes*. Compañías aseguradoras podrían encontrar esta aplicación de gran utilidad puesto que, en base a los atributos que definan a los clientes (edad, sexo, vehículo, número de accidentes previos, número de accidentes de los que ha sido responsable, etc.), podrían clasificarse estos en diferentes perfiles de riesgo y aplicar una tarifa u otra que sea más adecuada según las expectativas.
- *Clasificación de correos entrantes*. Otra de las aplicaciones posibles sería la de convertir el software en un clasificador de correos entrantes que, en base a los atributos de cada correo y las opciones especificadas por el usuario (remitentes prioritarios, horario laboral, proyecto en curso actual, etc.) -lo que implica que se estaría reciclando también el sistema de notificaciones con filtros-, podría mostrar los correos ordenados por interés (mayor interés primero) en la bandeja de entrada.

4. Bibliografía

1. https://slurm.schedmd.com/ 2018-03-09
2. https://www.r-project.org/ 2018-03-09
3. https://cran.r-project.org/web/packages/C50/C50.pdf 2018-03-09
4. https://es.wikipedia.org/wiki/Coeficiente_kappa_de_Cohen 2018-03-09
5. https://slurm.schedmd.com/sacct.html 2018-03-09
6. https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html 2018-03-09
7. https://docs.python.org/3.5/ 2018-03-09
8. https://www.java.com/es/about/ 2018-03-10
9. http://topepo.github.io/caret/index.html 2018-03-13
10. https://www.rulequest.com/see5-unix.html#BOOSTING 2018-03-13
11. https://anaconda.org/about 2018-05-01
12. https://es.wikipedia.org/wiki/Algoritmo_ID3 2018-05-01
13. https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/JRip 2018-05-01
14. http://weka.sourceforge.net/doc.packages/NNge/weka/classifiers/rules/NNge.html 2018-05-01
15. http://www.saedsayad.com/oner.htm 2018-05-01
16. http://weka.sourceforge.net/doc.packages/ridor/weka/classifiers/rules/Ridor.html 2018-05-01
17. https://github.com/twitter/AnomalyDetection 2018-05-03
18. Fábio M. Soares y Alan M.F. Souza (2016). *Neural Network Programming with Java*. Birmingham: Packt Publishing.
19. http://agilemanifesto.org/ 2018-05-03
20. https://es.wikipedia.org/wiki/Tiempo_Unix 2018-05-04
21. https://www.rulequest.com/see5-unix.html#RULES 2018-05-08
22. https://en.wikipedia.org/wiki/Medoid 2018-06-04

5. Anexos

1. Script de extracción de datos (*python*).
Ruta: anexos/extraccion/

2. Scripts de transformación de datos (*python* y *bash*)
Ruta: anexos/transformacion/

3. Script en R para la generación del modelo de clasificación.
Ruta: anexos/modelo/

4. Hoja de cálculo con la información de los experimentos.
Ruta: anexos/evaluation.ods

5. Archivos de código fuente del software final (*Java*).
Ruta: anexos/src/

6. Archivos de configuración del software final.
Ruta: anexos/configuracion/

- *config.cfg* : Archivo de configuración general.
- *data* : Directorio que representa la estructura de directorios utilizado por el software para funcionar con la configuración ofrecida.
- data/notifier/config.cfg : Archivo de configuración de notificaciones.
- *data/c50.R* : Script utilizado para generar los modelos de clasificación.
- *data/fix.sh* : Script utilizado por los scripts de recolección de datos para purgar los registros *CANCELLED*.
- *data/long/long_collection_script.sh* : Script de recolección de datos a largo plazo.
- *data/long/collections* : Directorio donde se exportan las recolecciones de datos a largo plazo.
- *data/short/short_collection_script.sh* : Script de recolección de datos a corto plazo.
- *data/short/collections* : Directorio donde se exportan las recolecciones de datos a corto plazo
- *data/predictor/data_to_predict_collection_script.sh* : Script de recolección de datos correspondientes a trabajos a predecir.
- *data/predictor/collections* : Directorio donde se exportan las recolecciones de datos correspondientes a trabajos a predecir.
- *data/predictor/predictions* : Directorio donde se exportan los datos de las predicciones realizadas.

7. Archivos necesarios para probar el sistema de consultas.
Ruta: anexos/consultas/

8. Análisis de resultados.
a) Análisis de predicciones.
Ruta: anexos/resultados/predictions_check
b) Análisis de notificaciones.
Ruta: anexos/resultados/notifications_check