Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

ÁREA DE PROYECTOS DE INGENIERÍA

TRABAJO FIN DE MÁSTER

METODOLOGÍA DE INTEGRACIÓN DE MODELOS BASADOS EN DATOS EN ENTORNOS DE PRODUCCIÓN

D. BUSTAMANTE ARIAS, LUIS
TUTOR: D. RODRÍGUEZ MONTEQUÍN, VICENTE

FECHA: JULIO DE 2018

Tabla de contenido

Índice de figuras

1.- Introducción

Los sistemas inteligentes permiten dar soluciones hoy en día a infinidad de problemas que hasta hace unos años las personas eran incapaces de resolver. Para el correcto desarrollo y explotación de dichos sistemas se necesitan conocimientos de distintas áreas tales como la inteligencia artificial, la cual se encarga del estudio y diseño de algoritmos capaces de tomar decisiones inteligentes, la ciencia de datos cuyo objetivo es el tratamiento, análisis y modelado de problemas, y la ingeniería de datos, capaz de diseñar arquitecturas e implementar sistemas capaces de ejecutar dichos modelos eficientemente y satisfacer la demanda de los usuarios. Con el objetivo de orientar y ayudar a los expertos de las diferentes áreas de conocimiento a cooperar y resolver los problemas que se les plantea a la hora del desarrollo de proyectos de modelado de datos, surge la necesidad de elaborar y aplicar metodologías que definan qué tareas se deben realizar en cada etapa del ciclo de vida del proyecto para su correcta ejecución.

En la actualidad, existen numerosas metodologías que describen qué hacer en las diferentes etapas de proyectos de modelado de datos, pero estas no describen cómo hacerlo ni que arquitecturas o tecnologías emplear para el correcto desempeño de las mismas.

El presente trabajo recoge una propuesta de metodología y arquitectura que sirva como guía para la correcta ejecución de proyectos de modelado de datos, concretamente acerca de modelos Machine Learning; subcampo de la inteligencia artificial encargada de la elaboración de modelos capaces de generalizar comportamientos a partir de una información de entrada. Para ello, se parte de una situación inicial, en la que se analizan los problemas metodológicos y tecnológicos que un equipo presenta a la hora de realizar este tipo de proyectos. Posteriormente se analizarán diferentes soluciones tecnológicas, se definirá una arquitectura y se propondrá una metodología que resuelva los problemas planteados acompañada de una implementación para ilustrar el proceso de desarrollo y su ciclo de vida.

2.- Alcance y situación actual

Como punto de partida para el desarrollo de este proyecto se parte de una situación en la que un equipo encargado de elaborar y ejecutar proyectos de modelado de datos presenta una serie de problemas a la hora de establecer una metodología en la realización de sus tareas. Concretamente es sabido que a la hora de diseñar las soluciones de puesta en producción de los modelos de Machine Learning no se tiene establecido un marco de tecnologías común que sirvan de referencia para todos los proyectos de pertenecientes al mismo ámbito.

A través de diversas entrevistas con los miembros del equipo se ha reunido la siguiente información:

- Las únicas tecnologías que se tienen definidas son aquellas relacionadas con el desarrollo de los modelos de Machine Learning. Se emplean Python y R, con una mayor tendencia de uso de Python sobre R siempre que sea posible; cuando determinadas librerías solo se encuentran disponibles en R se utiliza este frente a Python.
- Para permitir la comunicación entre los modelos y el usuario, se desarrollan interfaces específicas utilizando el framework .Net para la elaboración de aplicaciones de escritorio. Es sabido que un futuro se desea desarrollar, además, aplicaciones web para determinado fin.
- Como origen de datos para los proyectos, se utiliza un servidor que contiene archivos *csv* y *xlsx* con datos de diferentes proyectos y una base de datos MySQL la cual es accesible a los miembros del equipo para realizar los análisis y modelados oportunos.
- Los modelos se almacenan localmente junto con la aplicación de escritorio del usuario o esta accede a un servicio SOAP remoto que se encarga de servir el modelo.
- El equipo está comenzando a poner en producción sus modelos, hasta ahora solo se elaboraban para realizar análisis offline (sin tratar datos en vivo de los

usuarios). A pesar de no tener un gran número de modelos en producción ya se ha puesto de manifiesto la necesidad de definir una metodología y tecnologías que permitan:

  - Minimizar el tiempo de puesta en producción para agilizar el futuro despliegue de un mayor número de modelos.
  - Definir métricas de monitorización para los modelos de tipo supervisado. Se entiende por modelos de Machine Learning supervisados aquellos en los que en la fase de entrenamiento se proporcionan junto las variables de entrada, las salidas que corresponden a cada una de ellas.

    El objetivo que se pretende es poder visualizar la calidad de las predicciones que los modelos están enviando al usuario y si estas comienzan a degradarse, reentrenar los modelos con nuevos datos.
  - Evitar problemas de compatibilidad de librerías y dependencias entre unos modelos y otros. Cada uno debería de tener su propio entorno de entrenamiento independiente.

- Actualmente los modelos puestos en producción son reentrenados cada semana con una ventana temporal de seis meses. De cara a mejorar este aspecto, se desea establecer una comparativa de la calidad de los modelos entrenados inicialmente y que luego se han puesto en producción con aquellos que, o bien gracias a los datos en vivo que el usuario proporciona pueden estar reentrenándose de manera online, o bien, diferentes configuraciones de modelos más generalistas que puedan mejorar la calidad de los resultados ante las peticiones de los usuarios.

  Se desea establecer, además, un mecanismo de control de la degradación de la precisión de los modelos en producción, de tal manera que si la precisión de estos decaen de un cierto umbral se pueda tomar una decisión en base a las métricas de cada versión y seleccionar el de mayor calidad hasta el momento.
- El diseño de la nueva arquitectura tiene que permitir el despliegue de manera local, en la sede de la empresa. Por motivos de privacidad y confidencialidad de

los datos que se tratan en los análisis el despliegue en la nube no está permitido.

La Fig. 1 resume el estado actual del sistema que se está utilizando en producción. En ella se puede observar como actualmente los datos de producción son recabados por los entornos donde el usuario opera. Estos entornos o equipos de trabajo implementan las aplicaciones de escritorio que sirven como interfaz al modelo local o a un servicio SOAP remoto. No existe ningún sistema que albergue o controle a los modelos y para cada nuevo proyecto se ha de pensar, diseñar e implementar el modo en el cual estos se quieren poner en producción. Por último, el Data Scientist tiene que calcular manualmente la precisión de los modelos que se encarga de supervisar para que, si la calidad de estos decaen de un cierto umbral y el reentrenamiento no está mejorando los resultados, implementar otro tipo de modelo. En ocasiones, para esta tarea, la persona encargada tiene que desplazarse a la ubicación del entorno del usuario para aplicar los nuevos cambios.

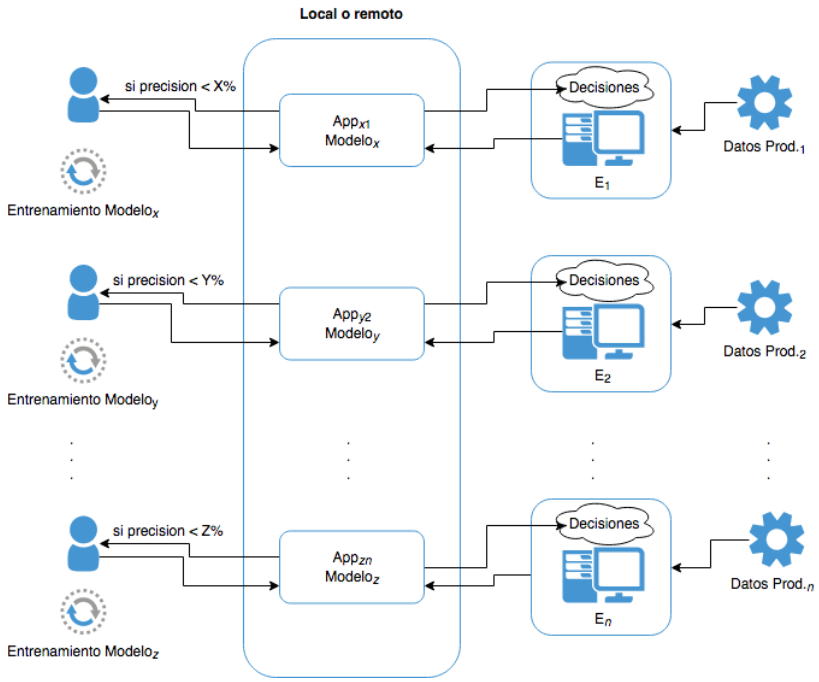


Figura 1. Diagrama de la situación actual

Teniendo en cuenta la información que se ha recabado del equipo de expertos, se desarrollará una propuesta de arquitectura que permita gestionar, de acuerdo a una metodología, las fases de análisis de datos, desarrollo de modelos de Machine Learning, implantación en producción, testing de nuevas configuraciones y monitorización de estos.

3.- Estado del arte de modelos R y Python en producción

En este apartado se ha realizado una búsqueda del estado del arte acerca de tecnologías que, por un lado, permitan desplegar plataformas de producción para el despliegue de modelos de Machine Learning. Por otro lado, permitan el desarrollo en lenguajes R y Python, siendo interesante explorar opciones que permitan interoperabilidades entre los lenguajes, es decir, utilizar funcionalidades propias de un determinado un lenguaje, desde otro, debido a que se tiene una mayor experiencia o preferencia por él o se quieren aprovechar determinadas funcionalidades específicas propias del lenguaje.

De acuerdo con el equipo de expertos es interesante analizar los siguientes puntos para cada alternativa: Requisitos software, tipo de licencia, lenguajes de programación que soporta, tipo de despliegue (Cloud o local), funcionalidades que aporta, como se implementan los modelos de Machine Learning y si permite monitorizar y actualizar dichos modelos.

3.1.- Plataformas Machine Learning

3.1.1.- Machine Learning Server

Es una plataforma dedicada a la analítica de datos, desarrollo de modelos de Machine Learning tanto en R como en Python y creación de interfaces REST con el objetivo de exponer las funcionalidades de los modelos desarrollados. Concretamente ofrece dos paquetes para facilitar la puesta en producción, “mrsdeploy” en R y “azureml-model-management-sdk” para Python. Su propietario es Microsoft la cual lanzó esta plataforma en septiembre de 2017.

Machine Learning Server es una tecnología que surge como evolución de R Server. Las novedades que incorpora esta frente su versión anterior es el soporte de Python en la plataforma; anteriormente solo permitía el desarrollo de modelos en R. Al igual que R Server, este debe de licenciarse como una característica suplementaria de

Microsoft SQL Server. En sistemas de producción se debe licenciar la plataforma (versión Enterprise) por core físico, lo que supondría un coste de 14,53$ dólares el núcleo.

ML Server se ejecuta sobre sistemas tales como Windows Server, Ubuntu y Apache Hadoop entre otros, permitiendo el despliegue en local o en el Cloud de Azure.

Respecto a las funcionalidades más destacadas que nos ofrece la plataforma podemos encontrar: Desde paquetes de Machine Learning compatibles con R y Python, que facilitan el desarrollo entre los diferentes programadores de ambos lenguajes, hasta la puesta en producción (proceso de transformación de modelos R o Python a servicios web que serán consumidos por aplicaciones de los usuarios). En cuanto al almacenamiento, las posibilidades de escoger diferentes tecnologías de BD viene muy limitado por Microsoft. Por defecto, la plataforma implementa SQLite; esta se puede sustituir por SQL Server en Windows o PostgreSQL en Linux.

La Fig. 2 muestra un esquema visual de los roles y operaciones que Machine Learning Server ofrece.

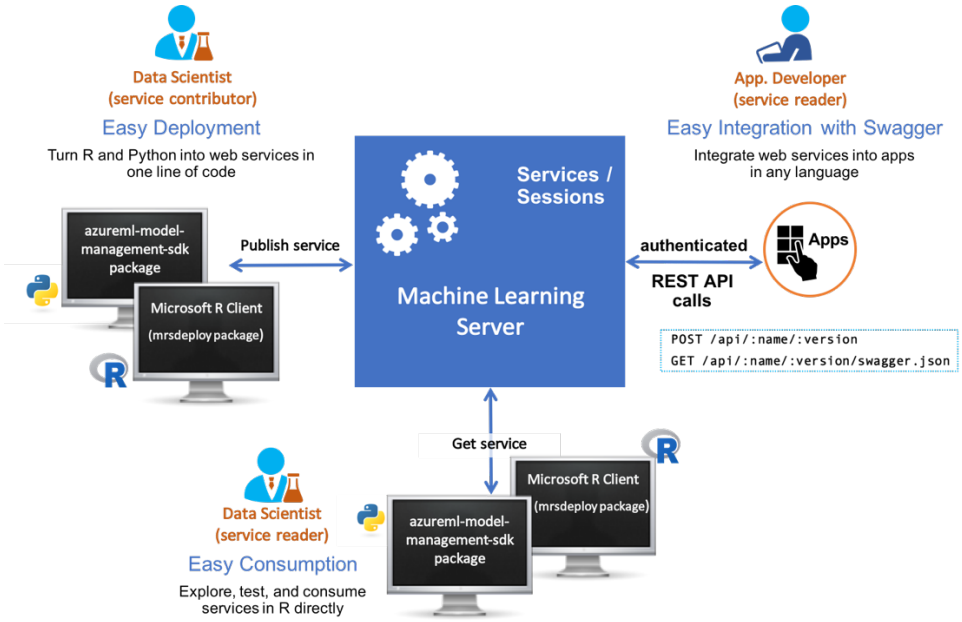


Figura 2. Machine Learning Server

En cuanto a los elementos que conforman la plataforma podemos agruparlos en:

- Web nodes: HTTP REST endpoints con los cuales el usuario puede interactuar directamente y hacer llamadas a la API definida para realizar la comunicación con los modelos. Este tipo de nodos se encargan además, de la gestión de acceso a la BD y envío de peticiones al Compute node para su procesamiento. Si se quiere aumentar la disponibilidad y evitar puntos únicos de fallo estos son los tipos de nodo que habría que replicar. Este tipo de nodos solo están disponibles en versiones Windows Server y versiones de Ubuntu LTS.
- Compute nodes: Nodos dedicados a la ejecución de código R y Python. Cada nodo tiene su propio pull de shells en ambos lenguajes y pueden ejecutar múltiples peticiones al mismo tiempo. Escalar este tipo de nodos supondría aumentar la capacidad de procesamiento de peticiones simultáneas y un aumento del balanceo de la carga. Este tipo de nodos solo están disponibles en versiones Windows Server y últimas versiones de Ubuntu LTS.

- Data Store: Referente a toda la parte de configuración de almacenamiento de datos.

Los tipos de configuraciones disponibles que se pueden desplegar con los elementos anteriormente descritos son:

One-box: Es la configuración mas simple. Esta pensada para realizar las primeras pruebas con el entorno, adaptarse a la plataforma y poner en producción los primeros modelos datos. Sin embargo, futuros desarrollos deberían realizarse con la configuración Enterprise debido a la necesidad de balancear la carga de usuarios.

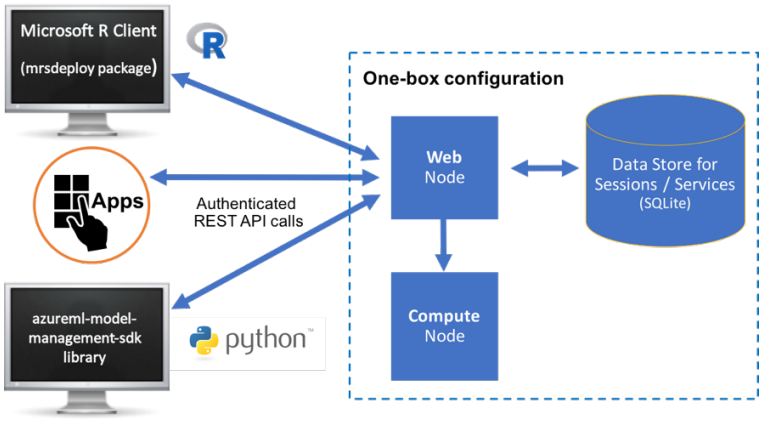


Figura 3. Configuración ML Server One box

Enterprise: Permite realizar escalado de nodos y autenticación de usuarios (LDAP o Azure Active Directory). Es la arquitectura empleada en las fases de producción.

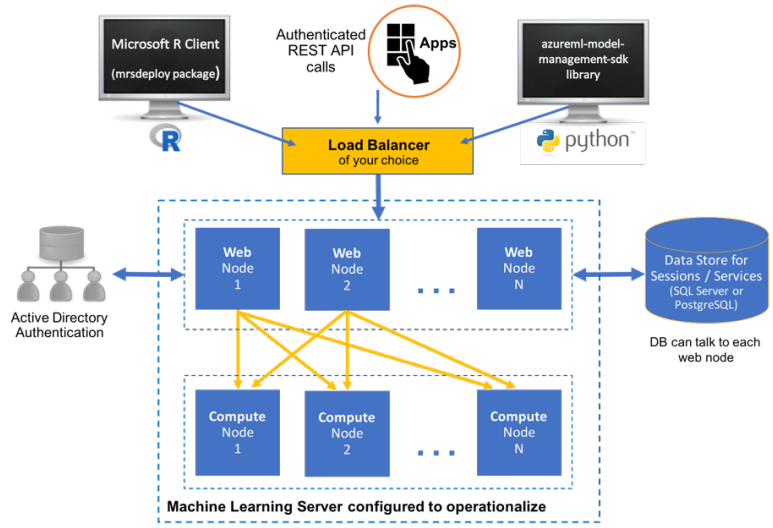


Figura 4. Configuración ML Server Enterprise

3.1.2.- H20

Es una plataforma de analítica predictiva y Machine Learning, desarrollada en Java, que permite por un lado entrenar y crear modelos sobre entornos Big Data, y por otro lado facilitar la puesta en producción de los mismos gracias a su modelo de escalado horizontal. H2O forma parte de una de las empresas "leaders" dentro del cuadrante mágico de Gartner referido a plataformas de Data Science y Machine Learning.

H2O facilita el modelado de problemas ofreciendo un conjunto de algoritmos de ML con su plataforma, entre los que destacan: GLM (para predicción de datos), algoritmos Deep Learning (para tratamiento de imágenes) y K-means y PCA (para agrupar y reducir la dimensionaldad de los datos). A través de su librería denominada *h2o* se pueden utilizar estos modelos tanto desde R como Python. Esta funcionalidad tiene como objetivo facilitar el uso de técnicas de Machine Learning a los usuarios que quieran ahorrar tiempo de investigación, búsqueda e implementación. Además, gracias a otra funcionalidad denominada "sparkling wáter" los modelos que ofrece h2o pueden ser ejecutados eficientemente en memoria y distribuidos sobre Spark, ofreciendo soporte a la librería MLib desde H2O.

En la Fig. 5 se pueden observar las partes que componen la pila de software de H2O. En el top de la pila se encuentran los clientes de las API´s REST para cada lenguaje que soporta H2O, aunque también es posible serializar los modelos a objetos tipo MOJO y POJO para su distribución.

En la parte baja de la pila se muestran diferentes componentes que se ejecutan dentro de un proceso JVM (Java Virtual Machine) el cual puede ser replicado entre los diferentes nodos de la arquitectura para aumentar la capacidad de procesamiento. Cada proceso JVM en H2O se componen de 3 capas: lenguaje, algoritmos e infraestructura core.

La capa de lenguaje implementa el Rapid Expression Evaluation Engine, el cual se encarga evaluar expresiones complejas sobre data frames mediante estructuras tipo árbol. El resultado de esta capa es utilizado por la capa de algoritmos la cual se encarga de la ejecución de estos; H2O también permite que el usuario use sus propios algoritmos aunque su ejecución no es tan eficiente.

Por último, cada proceso JVM implementa en su core la gestión de memoria y la gestión de CPU.

Todos los procesos JVM pueden acceder a bases de datos Spark, Hadoop y Standalone H2O.

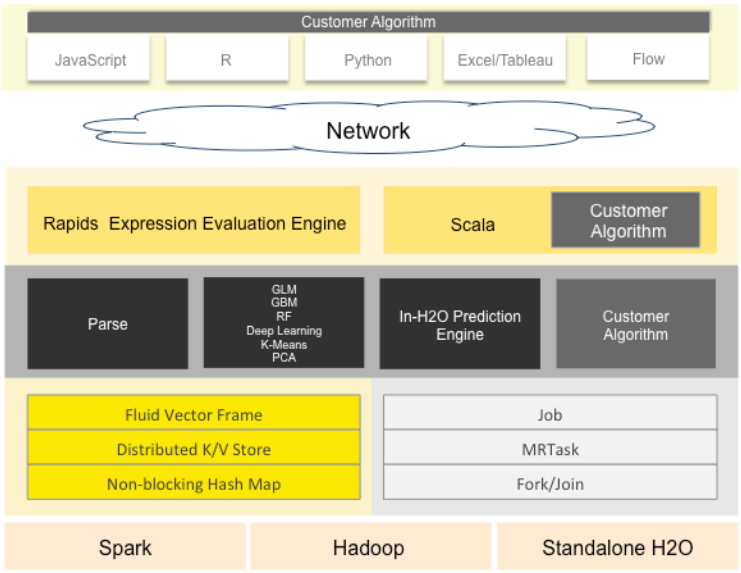


Figura 5. Arquitectura H2O

3.1.3.- Azure DataBricks

Plataforma desarrollada por la empresa Databricks, creada por fundadores Apache Spark junto con Microsoft. También existe la misma plataforma compatible con AWS, denominada Azure Databricks AWS. Puesto que ambas ofrecen las mismas funcionalidades se analizará la versión de Azure.

Es una plataforma de análisis basada en Apache Spark, optimizada para los servicios en la nube de Microsoft Azure. Proporciona flujos de trabajo optimizados y un área de trabajo interactiva que permite la colaboración entre científicos de datos, ingenieros de datos y analistas empresariales, orientados a trabajar con proyectos Big Data. Al estar alojada en el Cloud, los usuarios de Databricks se benefician de otras tecnologías complementarias como PowerBI para la visualización de datos o SQL Data Warehouse para el almacenamiento, además de la seguridad de los datos que proporciona el Cloud de Azure.

Ofrece un entorno para el desarrollo de modelos mediante notebooks interactivos a partir de una UI (También ofrecen API´s REST). En estos notebooks se puede programar en varios lenguajes simultáneamente gracias a diferentes etiquetas que permiten declarar zonas de código para R, Python, Scala y SQL. Sin embargo las variables entre diferentes zonas de código no se pueden compartir, son independientes. La Fig. 6 proporciona una imagen del flujo de trabajo que permite Azure Databricks.

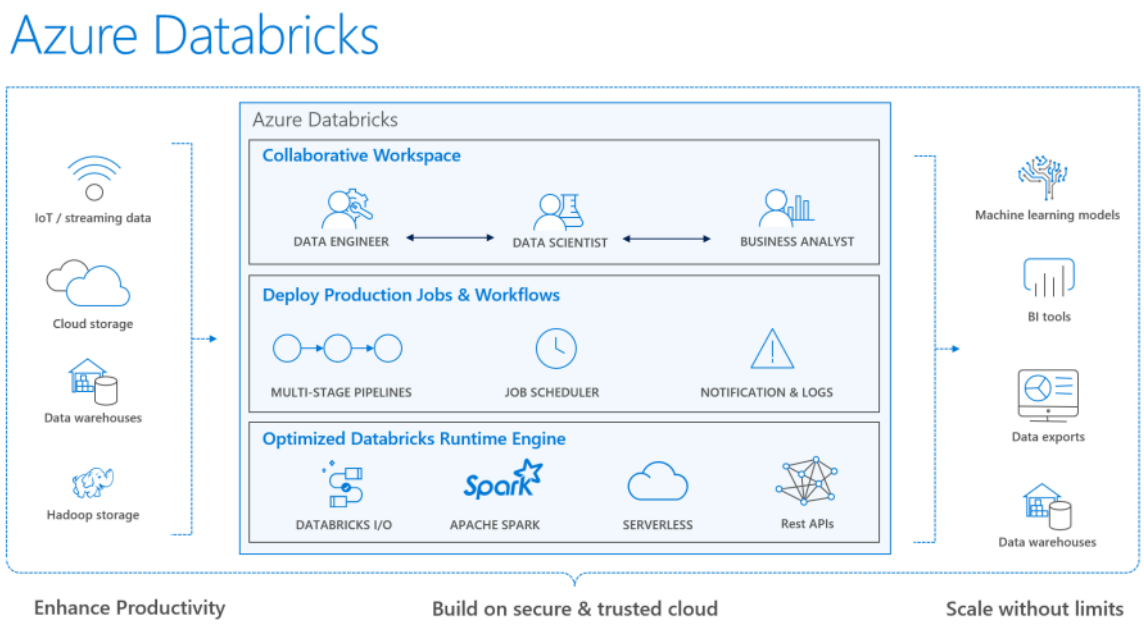


Figura 6. Flujos de trabajo Databricks

Dentro de la plataforma podemos resumir sus funcionalidades en:

- Clústeres de Spark completamente administrados por Azure.
- Área de trabajo para los usuarios interactiva de exploración y visualización.
- Funcionalidades de Spark:
    - SparkSQL: Módulo dedicado a trabajar con datos estructurados.
    - Streaming: Procesamiento y análisis en tiempo real para aplicaciones analíticas e interactivas.

- MLib: Biblioteca de Machine Learning que cuenta con los algoritmos y utilidades de aprendizaje más comunes, además de contar con primitivas de optimización más comunes.
- GraphX: Gráficos y cálculo gráfico para una amplia gama de casos de uso.
- Spark Core API: Incluye compatibilidad con R, Python, Scala y SQL.

3.1.4.- Azure Machine Learning Studio

Es un servicio en la nube de Azure que permite la creación de modelos de aprendizaje automático y soluciones de análisis predictivo a través de una interfaz web la cual permite arrastrar y soltar componentes para la construcción de un workflow de datos. Una vez se ejecuta el workflow Azure ML Studio origina un modelo entrenado con las especificaciones que se hayan seleccionado anteriormente. A continuación, este se puede poner en producción fácilmente a través de un servicio web.

A la hora de actualizar el modelo en producción, Azure ML Studio ofrece una serie opciones a través de su interfaz las cuales permiten:

- Cambio en la entrada o salida de los datos. Esta opción no modifica el modelo sino la forma en que son procesados los datos antes y/o después de realizar la predicción.
- Reciclar el modelo con datos nuevos. En esta opción se nos ofrecen dos nuevas posibilidades.
    - Reentrenar el modelo mientras se ejecuta el servicio web. Se mantienen las mismas variables pero las instancias son diferentes.
    - Entrenar un nuevo modelo usando diferentes datos. Las variables con las que el modelo es entrenado son diferentes.
    - Entrenar un modelo completamente diferente al que se encuentra en producción.

Cabe destacar que de todos los mecanismos de actualización que se ofrecen, ninguno permite una actualización automática, es decir, que este se reentrene cuando su precisión decae de un cierto umbral con los datos fijados por una ventana temporal. Queda en manos del usuario tomar la decisión de cuando y como estos se deben reentrenar a partir de métricas que el defina por su cuenta.

Por último, una característica reseñable es que el modelo a modo de servicio web que se despliegue en la nube puede ser monetizado. Permitiendo al usuario de Azure obtener un beneficio económico de sus clientes por usar sus servicios web.

3.1.5.- DeployR

DeployR es una tecnología de integración orientada al desarrollo de modelado analítico en R para web, escritorio, dispositivos móviles y dashboards. Por un lado facilita la conversión de los scripts R en servicios web analíticos y ofrece un servidor seguro para desplegar dichos servicios.

Es una tecnología de pago de la cual Microsoft es su propietario. Ha evolucionado de DeployR Open donde en sus comienzos era una herramienta Open-source perteneciente a la empresa Revolution Analytics. Puede ser complementada con la tecnología Machine Learning Server de Microsoft, teniendo que licenciar únicamente en este caso Machine Learning Server. De esta manera se posibilita tanto el despliegue en local como en Cloud.

DeployR divide a los desarrolladores que participan en su plataforma en 2 tipos de roles. Los Application Developers, usuarios que solo tienen conocimiento de las funciones expuestas por el servidor para realizar las llamadas adecuadas y obtener los resultados de los modelos de datos. Los Data Scientist que se encargan del desarrollo de los scripts referentes al análisis y modelado de datos, testing, asignación de permisos a los Application developers, permitiéndoles realizar llamadas a sus funciones desplegadas, y por último colaborar con estos para la adecuada

comunicación de sus modelos con la aplicación de usuario. En la Fig. 7 se puede observar el rol de cada uno dentro del workflow que presenta DeployR.

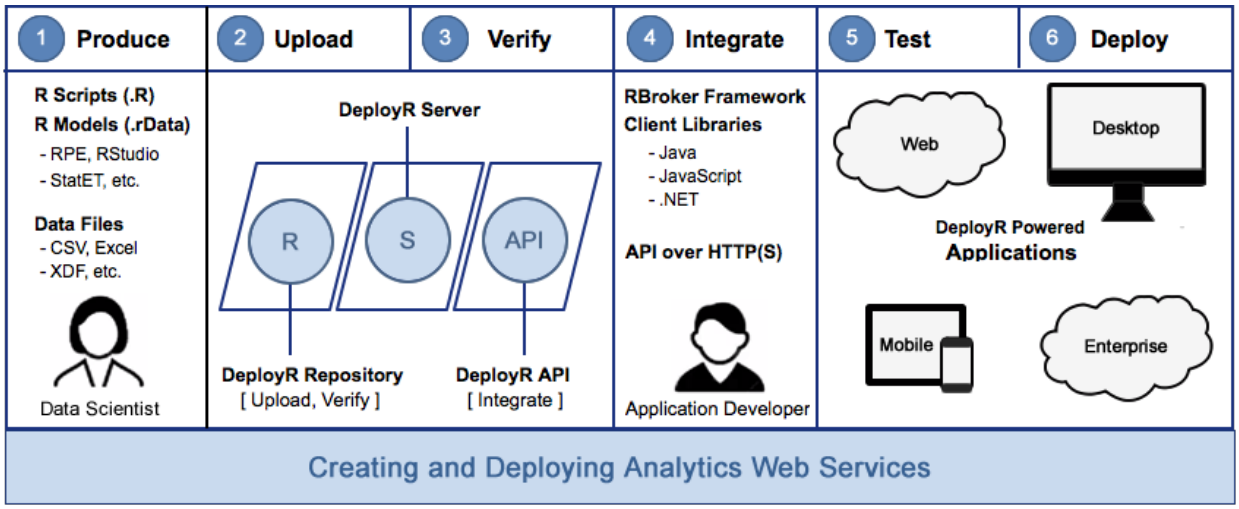


Figura 7. DeployR workflow

En resumen, el flujo de trabajo que permite DeployR sería el siguiente: el Data Scientist que ha desarrollado el modelo publica el script en el servidor DeployR. Una vez publicado, el script puede ser ejecutado por cualquier aplicación autorizada utilizando la API que DeployR proporciona y que lo expone como un servicio web. Por último el Application Developer puede obtener de la llamada a DeployR los datos devueltos por la función R que ha sido invocada a través de la API.

Se proporcionan librerías, para facilitar las comunicaciones con el servidor, para aplicación cliente en Java, Javascript y .NET aunque siempre se pueden usar llamadas a API de DeployR directamente.

DeployR ofrece dos tipos de versiones Enterprise y Open, las cuales son similares a las versiones que Machine Learning Server también ofrece. La diferencia entre ambas versiones reside en que la versión Enterprise permite replicar los nodos de cómputo y

aumentar el balanceo de carga entre diferentes servidores. Si se instala la versión Enterprise se requiere una dependencia que obliga a descargar una versión de R server (versión antigua de Machine Learning Server), por lo tanto estaríamos en el punto comentado anteriormente; se debe licenciar el producto como un producto Machine Learning Server igualmente.

3.1.6.- OpenCPU

Es una plataforma totalmente gratuita que permite exponer, a través de un servidor denominado OpenCPU Server ejecutado sobre Linux, una API HTTP para ejecutar scripts y funciones R, lo que resulta en un protocolo RPC clásico. El uso de OpenCPU permite separar los roles de Data Scientist y desarrollador de aplicaciones, ya que este último solo debe encargarse de realizar llamadas HTTP a OpenCPU para obtener los resultados de los modelos desde una aplicación web o de escritorio. Para este cometido OpenCPU ofrece una librería Javascript para facilitar las comunicaciones con OpenCPU Server.

Existen 2 versiones de OpenCPU para descargar:

- OpenCPU single-user: Se ejecuta dentro de una sesión interactiva de R. Su uso es para desarrollo local y posteriormente implantación en la versión cloud-server. Soporta una sola petición por cada instante de tiempo, ya que la sesión de R es single-thread.
- OpenCPU cloud-server: Es la versión OpenCPU utilizada para producción. Se ejecuta sobre Apache y NGINX. Permite la concurrencia entre usuarios, creando procesos de R a través de la técnica “fork” de Linux.

Si se desea realizar el análisis y entrenamiento remotamente, en el mismo servidor que OpenCPU Server, OpenCPU recomienda descargar RStudio Server, el cual permite la ejecución de código remoto y usar OpenCPU para hacer visible el código que se genere a través de las API´s.

El proceso de poner en producción un modelo es relativamente sencillo. Basta con encapsular las funcionalidades que deseamos servir en un paquete R, dejarlas en el directorio correspondiente que OpenCPU Server dedica para exponer las funcionalidades y este se encargará de hacerlo accesible.

3.1.7.- SAS Decision Manager

SAS es una empresa líder en software y servicios de analítica empresarial, ofrece hasta más de 150 productos y soluciones para ayudar a cubrir el ciclo de vida de proyectos relacionados con el modelado de datos. Se pueden encontrar soluciones que cubren los procesos desde la gestión de pequeños conjuntos de datos o Big Data hasta la visualización mediante dashboards interactivos de insights obtenidos por modelos de aprendizaje automático predefinidos que ellos mismos facilitan para la resolución de los problemas más comunes que se dan en el mundo empresarial. Todos los productos que se ofrecen forman de la arquitectura personalizada, que SAS oferta al cliente y es gestionable a través de un lenguaje de programación específico de SAS. El usuario puede elegir los componentes que el necesita de acuerdo con sus requisitos de negocio.

Debido a la gran cantidad de opciones que SAS ofrece, se analizará en concreto la denominada Decision Manager, puesto que cómo se explicará a continuación es la que cubre una gran parte de las necesidades que este proyecto pretende cubrir.

SAS Decision Manager es una plataforma que ayuda a la organizaciones a gestionar sus datos, reglas de negocio, modelos analíticos y workflows. A través de una interfaz de usuario es posible la gestión de las reglas de negocio, gestión de modelos, preparación de datos y despliegue.

El workflow de SAS Decision Manager comienza con la fase de Operational Data Preparation. El objetivo de esta fase es facilitar al usuario la comprensión y gestión de los datos de partida, ofreciendo mecanismos para construir diferentes tablas en las

que el usuario pueda almacenar y visualizar un resumen de los datos; la gestión de las tablas de datos se realiza a través de librearías de SAS.

La siguiente etapa consiste en el diseño y construcción de modelos predictivos. Se proporcionan mecanismos de ayuda en el entrenamiento de modelos y puntuación de los mismos. Una vez se finaliza el diseño estos pueden ser clasificados por el usuario, de acuerdo a sus puntuaciones, en “precisos”, modelos que detectan exitosamente patrones en el conjunto de datos de entrenamiento y producen el mínimo ratio de error en las predicciones y “robustos”, aquellos que serán fiables ante datos similares a los del conjunto de entrenamiento.

Posteriormente, de cara a la puesta en producción de los modelos previamente construidos, Decision Manager proporciona un complemento denominado Model Manager que ayuda a la gestión de modelos en producción a través del mismo interfaz gráfico, ofreciendo:

- Funciones de puntuación para observar el comportamiento de los modelo ante datos de testing, pudiendo estar probando simultáneamente numerosos modelos con distintas configuraciones ante el mismo problema.
- Métricas de rendimiento que facilitan la toma de decisión de la elección de qué modelos se ponen en producción.
- Monitorización de los modelos que se encuentran en producción y actualización de estos.
- Posibilidad de actualización automática de modelos cuando los resultados de las funciones de puntuación comienzan a degradarse.

Para finalizar, cabe destacar el elevado coste que tiene la adquisición de los productos de SAS. Alcanzando incluso costes de entre 100.000,00$ y 140.000,00$ por usuario al año.

3.1.8.- Google Cloud Machine Learning

Servicio Cloud ofrecido por Google que permite entrenar y desplegar modelos de aprendizaje automático y deep learning en una infraestructura optimizada para tal fin, con posibilidad de uso de GPU´s. El servicio también permite acceder a otros servicios de Google que se encargan de otras tareas del ciclo de vida de desarrollo, como es Google Cloud Dataflow para llevar a cabo las tareas previas al procesado de datos.

Los tipo de modelos soportado son los contenidos en la librería scikit-learn y el algortimo XGBoost. Además también se proporciona soporte para todos aquellos modelos de aprendizaje profundo desarrollados en Tensorflow.

Para llevar a cabo el desarrollo y análisis remoto se proporcionan notebooks que ofrece la tecnología denominada Jupyter.

3.1.9.- Amazon Sagemaker

Plataforma Cloud completamente administrada por Amazon que permite a los desarrolladores y Data Scientist crear, entrenar e implementar modelos de Machine Learning de manera rápida y simplificada. Facilita el acceso a los datos los cuales deben de estar albergados en Amazon S3 para su análisis y selección de cara a realizar el proceso de entrenamiento. Para la visualización de resultados SageMaker incluye notebooks de Jupyter hospedados en el Cloud y accesibles a través de un navegador, con el objetivo de facilitar el acceso a los resultados de los análisis.

Los lenguajes soportados son Python, R, Scala y Java.

Existen 2 opciones para gestionar el proceso de entrenamiento de los modelos:

- Entrenamiento con modelos predefinidos por SageMaker. Entre los que se encuentran: Linear Learner, Factorization Machines, XGBoost, K-means y PCA, entre otros.

- Uso de Docker para el despliegue del modelo personalizado en SageMaker. En esta opción el desarrollador debe encargarse de que el contenedor exponga las funcionalidades de entrenamiento e inferencia del modelo a través de una API REST.

En ambas opciones toda la infraestructura subyacente es administrada automáticamente y puede escalar fácilmente.

Finalizado todo el proceso de puesta en producción, SageMaker se encarga de hacer accesibles los modelos a través de puntos de enlace HTTPS los cuales pueden ser llamados desde cualquier aplicación.

3.2.- Tecnologías orientadas a la interoperabilidad de modelos

Como se ha comentado en el apartado de referente situación inicial, actualmente existen diversas dificultades de interoperabilidad entre los desarrolladores debido a que determinados modelos se desarrollan en Python y otros en R. En esta sección se proponen una serie de tecnologías con el objetivo de mejorar la interoperabilidad de los desarrollos. Por un lado se verán técnicas dedicadas a la serialización de modelos y por otro lado se mencionaran que tecnologías permiten crear API´s para permitir el acceso a los modelos Python y R. Estas tecnologías pueden ser usadas dentro de cualquier plataforma de Machine Learning que admite R y Python como lenguajes de programación. Más adelante, en el apartado dedicado a la propuesta de arquitectura, se explicarán cuales y como se utilizarán dentro del sistema propuesto.

3.2.1.- PMML

Es un lenguaje de marcado de texto XML utilizado para representar modelos de minería de datos. Fue creado por la organización Data Mining Group (DMG), la cual se compone de un conjunto de empresas, de las que forman parte IBM y SAS, con el objetivo de proporcionar un estándar común para la serialización de modelos

predictivos el cual pueda ser leído y procesado por cualquier lenguaje que soporte este estándar.

Existen librerías tanto en R (pmml) como en Python (Py2PMML) para leer y serializar determinados tipos de modelos al lenguaje PMML.

Los modelos soportados por el paquete pmml en R son:

- ksvm (kernlab): Máquinas de Vectores Soporte.
- nnet: Redes neuronales.
- rpart: C&RT Árboles de decisión.
- lm & glm (stats): Modelos de regresión lineal y binaria.
- arules: Reglas de asociación.
- kmeans and hclust: Modelos de clustering.
- multinom (nnet): Modelos de regresión logística multinomial.
- glm (stats): Modelos lineales generalizados para clasificación y regresión.
- randomForest: Modelo de Random Forest para clasificación y regresión.
- naiveBayes (e1071): Clasificador Naive Bayes.
- glmnet: Modelo de regression lineal basado en ElasticNet.
- ada: Versión estocástica del algoritmo gradiente descendiente.
- svm (e1071): Máquinas de Vectores Soporte.

En Python existe la posibilidad de exportar determinados modelos de la librería scikit-learn, las clases soportadas son:

- RandomForestClassifier: Modelos Random Forest.
- RandomForestRegressor: Modelos Random Forest.
- DecisionTreeClassifier: Árboles de decisión.
- DecisionTreeRegressor: Árboles de decisión.
- KMeans: Clustering
- LogisticRegression: Regresión Logística.

- LinearRegression: Regresión Lineal.
- GaussianNB: Naive Bayes Gaussiano.
- BernouilliNB: Naive Bayes de Bernoulli.

Como se ha explicado PMML permite la interoperabilidad de modelos de Machine Learning entre diferentes lenguajes entre los que se encuentran R y Python. A pesar de ello, actualmente el número de modelos soportados está muy restringido y no permite la serialización de modelos personalizados creados por el usuario. Lo que hace que el uso de PMML sea inviable para la solución propuesta.

3.2.2.- Flask

Es un microframework web. Flask junto a Django son los frameworks web más utilizados en Python. La diferencia entre uno y otro reside en que Flask se denomina "micro" puesto que trata de mantener el core de la aplicación simple pero extensible. Esto quiere decir que, a diferencia de Django que implementa toda su funcionalidad en el core, Flask lo hace a través de extensiones que se pueden ir añadiendo a la aplicación según se requiera. Puesto que el objetivo que se busca con las tecnologías de interoperabilidad es exponer únicamente determinadas funcionalidades de los modelos predictivos a través de API´s REST, se opta por explicar Flask frente a Django puesto que este es más ligero y flexible.

Al ser Flask un microframework web genérico, este no implementa ninguna función en particular para trabajar con modelos de Machine Learning. La idea de uso de Flask en esta propuesta es permitir a los Data Scientist exponer las funcionalidades que ellos consideren necesarias a través de las API´s REST y estas hacerlas accesibles a las aplicaciones de usuario. Como se puede observar en la Fig. 8, las funciones más comunes que se exponen son las relacionadas con el entrenamiento del modelo y predicción de resultados a partir de datos nuevos (inferencia).

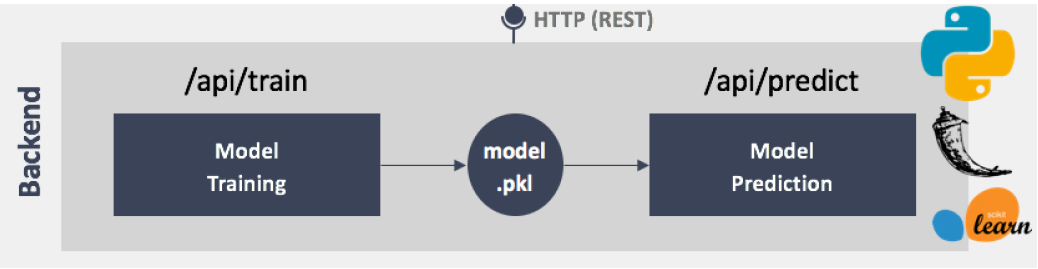


Figura 8. Flask REST API

Como los modelos deben estar previamente entrenados, estos deben ser serializados una vez finalice su fase de entrenamiento; la función de predicción se encargará de de-serializar el objeto, obtener los datos en formato JSON, realizar la inferencia y devolver el resultado a la aplicación del cliente en formato JSON.

Respecto a la serialización, se propone el módulo Pickle de Python. Este implementa los protocolos binarios necesarios para la serialización y de-serialización de objetos. Una precaución a tener en cuenta es que si serializan modelos construidos con librerías como por ejemplo scikit-learn, la versión de la librería debe ser la misma cuando se serialize un modelo y se de-serialize; se debe mantener la consistencia y fijar la versión con la que se va a desarrollar el modelo.

3.2.3.- Plumber

Es una librería de R que pretende solucionar los mismos problemas que se buscaban resolver con el microframework Flask; simplificar la creación de API´s REST para exponer las funcionalidades de los modelos de datos, en este caso para los modelos desarrollados en R. Se puede considerar como una versión más ligera que la plataforma vista con anterioridad OpenCPU. OpenCPU pretendía facilitar la creación de API´s REST y para ello proporcionaba una plataforma basada en NGINX y Apache para la gestión centralizada de los modelos programados en R y su puesta en producción; Plumber elimina la parte referente al servidor y proporciona únicamente los mecanismos para definir las API´s REST mediante un tipo especial de

comentarios. La tecnología que se utilice para el hosting de las aplicaciones queda de mano del usuario. Se recomienda el uso de Docker o RStudio Connect.

Al igual que ocurría con OpenCPU, Plumber sufre el problema de que R es single-thread. Por lo tanto, una sola instancia de Plumber solo es capaz de atender una petición de usuario de cada vez. La solución a este problema, para los entornos de producción, está en la creación de un conjunto de instancias de R que proporcionen respuesta simultánea a múltiples peticiones y un balanceador de carga que se encargue de la distribución de las cargas sobre diferentes instancias de R.

3.2.4.- RPY2

Otro tipo de tecnologías de interoperabilidad entre modelos escritos en diferentes lenguajes son aquellas que nos permiten ejecutar código o utilizar librerías de un lenguaje determinado desde otro. Un caso típico es el uso de este tipo de librerías por parte de desarrolladores con conocimientos acerca de un lenguaje que necesitan acceder a funciones escritas en otro lenguaje.

RPY2 es una librería de Python que permite servir de interfaz con el lenguaje R, proporcionando el beneficio de las librerías y funciones que este ofrece mientras el proyecto es desarrollado en el lenguaje nativo Python. El acceso a la funcionalidades de R se realiza a través de la C-API que este proporciona. Se puede resumir la funcionalidad de RPY2 en:

- Una interfaz de alto nivel que permite el uso de las funciones de R como si fuesen objetos en Python, proporciona una conversión similar a estructuras de datos como numpy y pandas y permite el uso de ciertas librerías gráficas de R, como ggplot, desde el propio lenguaje nativo.
- Una interfaz de bajo nivel muy cercana a la C-API.

3.2.5.- Reticulate

Proporciona las mismas funcionalidades que RPY2 pero en este caso utilizando como lenguaje nativo R. Reticulate permite la ejecución de un intérprete de Python dentro de la propia sesión de R. Las opciones de interacción con la sesión son:

- Uso de la sesión Python dentro de R Markdown: Permite una comunicación bi-direccional entre R y Python dentro de los documentos tipo markdown.
- Importar módulos de Python: Gracias a la función import de Reticulate se puede importar cualquier módulo de Python y llamar a sus funciones como si se tratase de un objeto R.
- Sourcing Python scripts: La función sourcing tiene la misma finalidad que en R, al aplicar la función source sobre un script de Python todas las funciones y objetos pasan a estar disponibles a la sesión de R.
- Python REPL: Crea una consola interactiva en Python dentro de R. Los objetos que se creen dentro de esa consola de Python pasan a estar disponibles a la sesión de R.

Cuando se retornan valores desde Python a R estos son convertidos de nuevo a tipos de datos en R. Si se retorna un objeto personalizado desde Python a R, lo que se devuelve es una referencia a ese objeto. Se pueden realizar llamadas a los métodos y acceder a sus propiedades como si fuera una instancia de una clase de R.

4.- Evaluación de alternativas

En este apartado se analizarán las utilidades y carencias que presentan las plataformas vistas en el estudio del estado del arte de modelos en producción visto anteriormente, respecto a las requisitos de los que se parte en este proyecto.

- Machine Learning Server: Permite el despliegue local con distintas configuraciones para construir una plataforma de ML, además de soportar los lenguajes de programación R y Python. Sin embargo, al no ser una tecnología Open-source necesita ser licenciada, concretamente esta debe licenciarse junto con una versión de SQL Server lo que limita la libertad de elección de la base de datos a utilizar, por ejemplo, si se requiriera una base de datos NoSQL esta plataforma no sería factible. Por último, la plataforma no implementa ningún control del rendimiento los modelos en producción, ni forma de reentrenarlos y actualizarlos cuando la precisión de estos comienza a degradarse.
- H2O: La plataforma facilita el proceso de puesta en producción de modelos de Machine Learning que ellos mismos te proporcionan. Sin embargo, la plataforma está orientada al tratamiento de Big Data, puesto que las bases de datos que esta soporta están orientadas al procesamiento de grandes cantidades de datos; las necesidades de este proyecto no requieren de tales plataformas. Por otro lado, al serializar los modelos en objetos Java pueden surgir dificultades a la hora de importar estos en las aplicaciones .NET actuales de los usuarios.
- Azure Databricks: Proporciona un entorno de trabajo para la colaboración entre los diferentes roles involucrados en un proyecto de modelado de datos, además del procesamiento en la nube y acceso a múltiples herramientas de Spark. El problema de esta tecnología es que no permite el despliegue local, solo en el Cloud de Azure además de ser una herramienta totalmente de pago a pesar de utilizar numerosas herramientas Open-source de Spark.
- Azure Machine Learning: Plataforma en la nube que facilita el esfuerzo de desarrollar modelos de datos y realizar análisis mediante una interfaz que actúa

a modo de lienzo interactivo. Está más enfocada a usuarios que desconocen los lenguajes de programación que a usuarios avanzados de R y Python. Esta tecnología es la única, de las que se han analizado que ofrece mecanismos para actualizar los modelos puestos en producción. Por el contrario, presenta los inconvenientes de que no permite despliegue local, ni ofrece métricas para monitorizar los modelos en producción.

- DeployR: Solo permite soporte para trabajar con modelos desarrollados en R. Es una tecnología que ha tenido su auge en el pasado cuando era Open-source, actualmente debido a que es una herramienta de pago, debe licenciarse con Machine Learning Server y no aporta novedades respecto otras tecnologías Open-Source que ofrecen características similares; ha entrado en desuso.
- OpenCPU: Permite la puesta en producción local de modelos desarrollados en R. Facilita la creación de API´s y el procesamiento de datos distribuido, liberando carga de computación del equipo del usuario. Pero no soporta Python ni el ofrece métricas de rendimiento ni opciones de actualización de los modelos.
- SAS Decision Manager: Es una herramienta que facilita la construcción de workflow a lo largo del ciclo de vida de un proyecto de modelado de datos. Además de ser la única herramienta que proporciona mecanismos de métricas de rendimiento y actualización de modelos para los sistemas en producción. Sin, embargo no ofrece compatibilidad alguna con los lenguajes Python y R sino que la gestión de cada etapa se puede personalizar a través del lenguaje de programación de SAS. Esto conlleva a una formación extra de todos los miembros del equipo que vayan a hacer uso de SAS y el abandono de los lenguajes con los que ellos trabajaban hasta el momento. Lo que hace de SAS una herramienta muy costosa en cuanto a formación, además de adquisición.
- Google Cloud Machine Learning: Servicio Cloud que se encarga de cubrir las fases de entrenamiento y puesta en producción de determinados modelos de Machine Learning ofrecidos por la librearía scikit-learn o el algoritmo XGBoost,

además de proporcionar soporte a todo los modelos Deep Learning desarrollados con la librearía de TensorFlow. No permite despliegue local, ni soporte para modelos personalizados que el usuario desarrollo en lenguajes como R y Python. Además, para cubrir determinadas fases del ciclo de vida de desarrollo es necesario recurrir a otros servicios del Cloud de Google. Tampoco ofrece métricas de rendimiento y mecanismos de actualización de los modelos. En definitiva Google Cloud Machine Learning es un servicio orientado más a dar capacidad de cómputo para modelos Deep Learning que a servir como plataforma de soporte para todo tipo de algoritmos de Machine Learning.

- Amazon SageMaker: En resumen, la plataforma Cloud de SageMaker tiene como objetivo facilitar el workflow de acceso a datos, entrenamiento y puesta en producción de modelo de datos. Ofrece, además, algoritmos predefinidos de cara simplificar el modelado por parte de usuarios sin conocimientos de programación. A pesar de ello, esta plataforma no permite el despliegue local.

Como se ha podido comprobar existen numerosas plataformas que ofrecen las mismas prestaciones a los usuarios, como resumen de las alternativas vistas hasta el momento, se presenta la Tabla 1 donde, para cada plataforma, se verifica si cumple con los requisitos que se precisan para satisfacer las condiciones iniciales que los usuarios presentan. Se valora: si la plataforma es de pago u Open-source, si permite un despliegue en servidores locales o solo en la nube, si soporta los dos principales lenguajes de programación usados en el desarrollo de los modelos, si la plataforma implementa métricas para medir el rendimiento de los modelos que se encuentren en producción y si esta implementa algún mecanismo que permita actualizar de manera automática los modelos cuando las calidades de las predicciones se degraden.

(*): La plataforma cumple la característica.

Alternativas	Sin coste	Despliegue local	Soporte R	Soporte Python	Métricas de rendimiento	Actualización de modelos
ML Server		*	*	*		
H2O	*	*	*	*		
Azure Databricks			*	*		
Azure ML Studio			*	*		*
DeployR		*	*			
OpenCPU	*	*	*			
SAS Decision Manager		*			*	*
Google Cloud ML						
Amazon SageMaker			*	*		

Tabla 1. Comparativa plataformas ML

Como se puede observar actualmente no existe una plataforma en el mercado que cubra los requisitos definidos en el alcance del proyecto; ya sea una tecnología con coste o sin coste. También se han descartado aquellas plataformas que están orientadas al tratamiento de Big Data y aquellas que únicamente pueden ser

desplegadas en el Cloud, ya que uno de los requisitos fijados en el alcance era que la plataforma debía permitir el despliegue local en las instalaciones del usuario.

Por este motivo se va optar por la propuesta de una arquitectura y una metodología que permitan la definición de un flujo de trabajo único para los proyectos de modelado de datos, así como las tecnologías necesarias para que pueda realizarse cada uno de los pasos implícitos dentro del ciclo de vida del proyecto.

La propuesta contemplará el uso de gran parte de las tecnologías de interoperabilidad definidas anteriormente, así como otras nuevas que permitan la elaboración de una plataforma de cómputo para los proyectos de Machine Learning.

5.- Propuesta de arquitectura

Realizado el estudio acerca del estado del arte de plataformas de Machine Learning y explicado los motivos por los cuales ninguna plataforma cubría los requisitos definidos en el alcance del proyecto, se optará por el diseño de una arquitectura a medida, la cual cubra y de soporte a cada uno de los puntos del ciclo de vida de los proyectos de modelado de datos.

El tipo de arquitectura elegida estará orientada a microservicios. Este diseño permite descomponer las diferentes funcionalidades que conlleva la ejecución de un proyecto de modelado de datos en pequeños servicios cuya comunicación se realiza a través de mecanismos ligeros. Estos servicios serán gestionados y encapsulados a través de contenedores, que como se explicará más adelante, proporcionarán numerosos beneficios.

Como ejemplo práctico, para explicar que componentes pueden conformar una arquitectura basada en microservicios, se expondrá el caso de la arquitectura denominada Michelangelo; empleada por Uber para gestionar todos los modelos de Machine Learning que sirven predicciones a sus servicios.

Al igual que las necesidades planteadas en el alcance de este proyecto, Uber partió de una condición similar. Por un lado, los Data Scientist desarrollaban los modelos de Machine Learning en múltiples lenguajes lo que daba lugar a problemas de interoperabilidad y comunicación, por otro lado los ingenieros de datos tenían que encargarse de diseñar y construir sistemas específicos para poner en producción cada proyecto. No existían flujos de trabajo definidos para la creación y gestión del entrenamiento y predicción de datos. Los entrenamiento de los modelos se realizaban en los equipos locales de los Data Scientist, lo que estaban limitados a la capacidad de cómputo sus propios PCs. Debido a estas necesidades Uber comenzó a desarrollar la plataforma Michelangelo.

El esquema de las partes que componen Michelangelo se pueden ver en la Fig. 9.

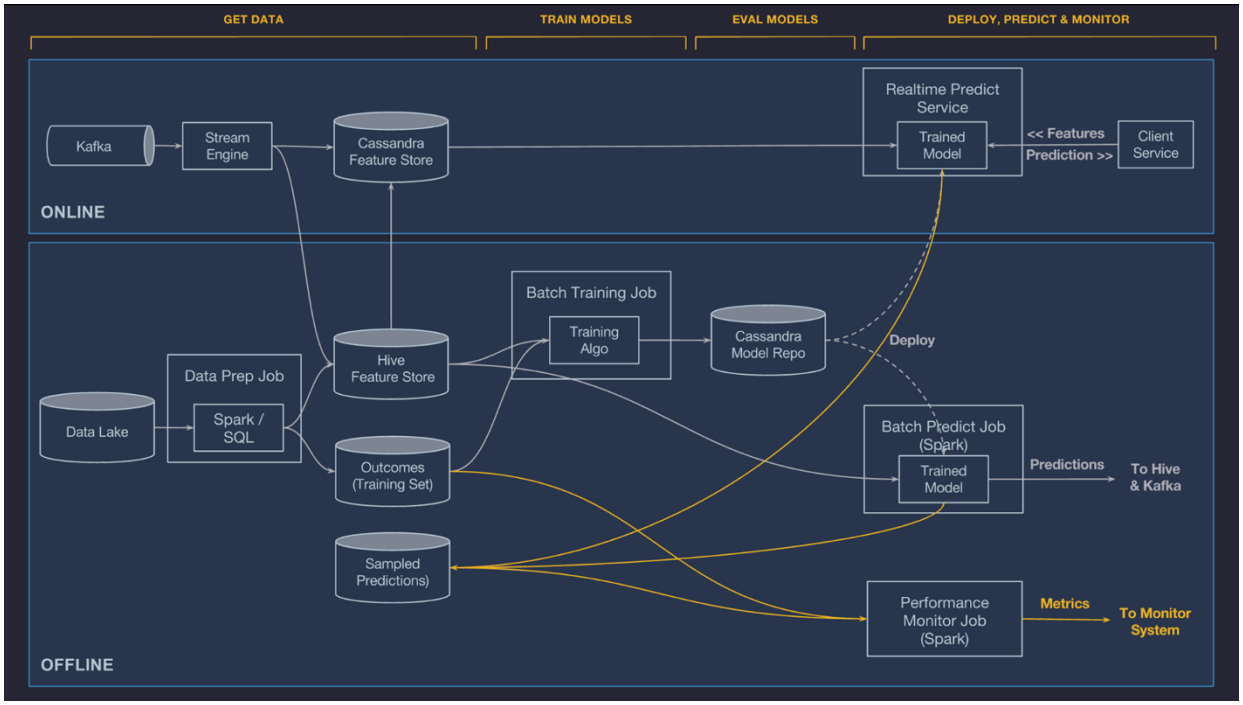


Figura 9. Arquitectura Michelangelo

La arquitectura pretende dar soporte a los siguientes pasos del flujo de trabajo de Uber:

1. Obtención de datos

   Esta paso está dividida en dos flujos: online y offline. La parte offline recoge todas las transacciones y logs que se realizan en las aplicaciones de Uber, posteriormente estas son almacenadas en un Data Lake constituido por el sistema HDFS de almacenamiento de Hadoop y accesible tanto desde Spark como desde Hive SQL.

   La parte online se encarga de dar predicciones en tiempo real a los usuarios, por lo que el acceso a Hadoop no es posible, por ello los datos de los modelos que realizan la inferencia online son almacenados en una base de datos Cassandra debido a su baja latencia. Esta base de datos se alimenta de dos fuentes: de un stream de datos recogidos online a través de Kafka, la cual es

una plataforma encargada del procesamiento de flujos de datos. Y de datos históricos almacenados en Hadoop la cual transfiere parte de su contenido a Cassandra cada cierto período de tiempo.

2. Entrenamiento de modelos.

   Se realiza a través de un navegador web o API utilizando notebooks Jupyter, los cuales proporcionan una interfaz para la ejecución remota de código. Una vez se finaliza esta fase las variables de modelos son introducidos en una base de datos Cassandra.

3. Evaluación de modelos.

   Cuando se finaliza fase de entrenamiento toda la información referente a la importancia de variables, algoritmos y ajuste de hiper-parámetros son almacenados. Michelangelo proporciona a través de un dashboard de monitorización medidas acerca de la precisión de los modelos, visualizaciones de árboles de decisión e informes acerca de las distribuciones de las variables.

4. Despliegue de modelos en producción.

   El despliegue de los modelos se produce mediante microservicios. Los contenedores que encapsulan los servicios de producción se encargan de acceder a los sistemas que albergan los modelos, cargarlos en memoria y posteriormente esperar las peticiones para realizar la inferencia.

5. Inferencia.

   Una vez los modelos son desplegados y cargados en memoria por el contenedor que se encarga de servirlos, estos son usados para realizar predicciones basadas en datos que, o bien vienen de un pipeline offline interno y las respuestas se almacenan en la BD Hive, o bien directamente los resultados del servicio online de un cliente.

6. Monitorización del rendimiento de los modelos.

   Para asegurar que un modelo funcione correctamente en el futuro, es crítico monitorizar la calidad de sus predicciones. Para llevar a cabo esta tarea Michelangelo almacena un porcentaje de las predicciones para posteriormente cruzarlas con las observaciones reales. Con esta información, se pueden

generar métricas de rendimiento en vivo acerca de las predicciones de los modelos. Toda esta información es servida a través una aplicación de gestión que sirve una interfaz web y una API, de tal manera que se pueda integrar con el sistema de monitorización de Uber y la infraestructura de alerta.

Al igual que el caso de Uber, existen otros muchos ejemplos de grandes compañías que adoptan este tipo de arquitecturas para gestionar y poner sus servicios en producción. Entre ellas se encuentran: Amazon, Netflix, Twitter, Paypal, Mango, etc..

Viendo la importancia, flexibilidad y utilidad de las arquitecturas orientadas a microservicios. A continuación, se expondrán un conjunto de tecnologías que se utilizarán para el diseñe e implementación de la propuesta de arquitectura. Finalmente, en este apartado, se verá como se relacionarán cada una de las tecnologías entre sí para dar soporte a todos los servicios que se requieren.

5.1.- Definición de componentes

5.1.1.- Docker

La idea de una arquitectura basada en microservicios se basa que cada componente de la plataforma o aplicación sea un pequeño servicio independiente que posee mecanismo de comunicación ligero para comunicarse con el resto de servicios que componen la arquitectura. Como se mencionó en la introducción de la propuesta, se pretende utilizar contenedores para llevar a cabo la construcción de cada microservicio.

Los contenedores son un método de virtualización que proporciona el sistema operativo el cual permiten ejecutar una aplicación y sus dependencias en procesos totalmente aislados. Esto permite empaquetar con facilidad el código de una aplicación, su configuración y dependencias, lo que se traduce en un aumento de la productividad para los desarrolladores y gestión de control de versiones. La Fig. 10,

ilustra, de forma gráfica, la diferencia entre este método de virtualización y las máquinas virtuales.

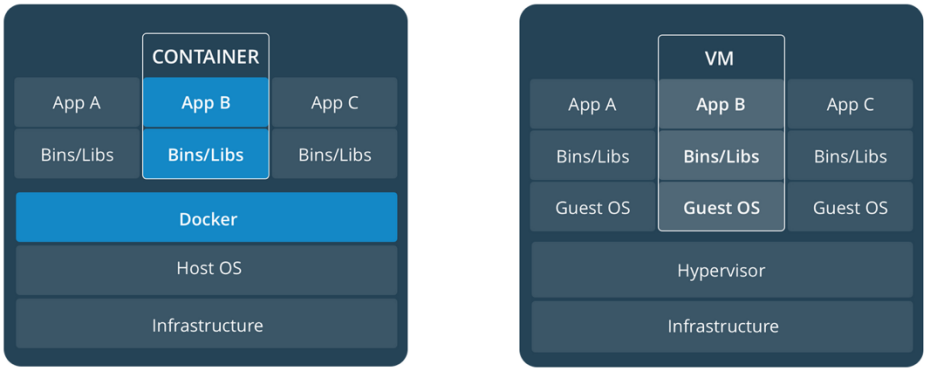


Figura 10. Contenedores y MV

Los contenedores y las máquinas virtuales persiguen objetivos similares; facilitar el despliegue y aislamiento de recursos. Sin embargo, a nivel de funcionalidad son diferentes, ya que los contenedores realizan la virtualización sobre nivel del S.O anfitrión, no de hardware, lo que se traduce en que estos sean más portables, debido a que su ejecución es prácticamente instantánea, y eficientes ya que consumen menos espacio en disco.

Al igual que ocurre con las MV, los contenedores requieren de un software que gestione su ciclo de vida (creación, gestión, eliminación…). Es por ello que se utilizará Docker para llevar a cabo esta tarea. Docker es una plataforma de código abierto que proporciona diferentes productos, como Docker Engine y Docker Swarm para realizar la gestión de contenedores y puesta en producción respectivamente. Concretamente se utilizará la versión Community la cual no tiene coste alguno. Existe otra versión, Enterprise que añade el soporte de Docker a un rango mayor de S.O, proporciona dashboards que muestran el consumo de recursos por parte de los contenedores, etc… Para las necesidades que se pretenden cubrir en este proyecto no es necesario recurrir a la versión de Enterprise.

5.1.1.1.- Docker Engine

Docker Engine es el producto central de Docker. Se encarga de la creación y la ejecución de contenedores en Docker a partir de una imagen. Una vez se tiene un contenedor en ejecución, todas las modificaciones que se hagan en este desaparecerán si es eliminado; al volver a ejecutar la imagen que dio lugar al contenedor se volvería a su estado inicial. Existen dos técnicas para salvar las modificaciones que se realicen en un contenedor desde su creación: mediante el uso de un comando commit, el cual da lugar a una nueva imagen con los cambios realizados hasta el momento o mediante un volumen lógico montado sobre un directorio del sistema anfitrión que se incorpora al sistema de ficheros del contenedor en el cual se almacenen los datos y cambios realizados.

Una imagen de Docker es un archivo en el que se define que servicio o programa se va a ejecutar sobre S.O del host. Todos las imágenes parten de una versión de un S.O Linux, a partir de la cual se van definiendo colecciones de ordenes, como por ejemplo cambios sobre el sistema de ficheros del contenedor, instalación de librerías o dependencias software, etc… de tal manera, que cuando Docker Engine ejecute la imagen para dar lugar a un contenedor en ejecución, este contendrá las modificaciones definidas en el archivo de la imagen que lo originó; este archivo se denomina Dockerfile. Para ilustrar brevemente el ciclo de vida de un contenedor se proporciona Fig. 11.

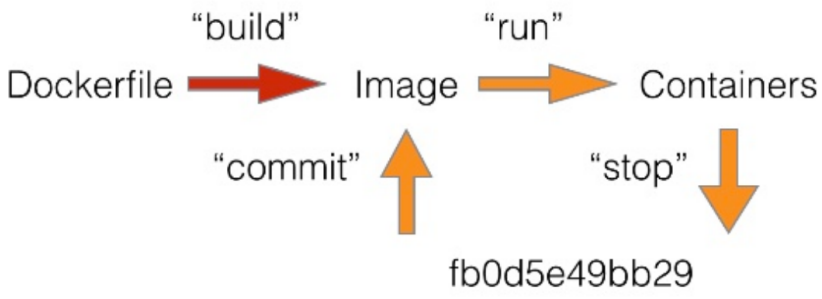


Figura 11. Ciclo de vida de un contenedor Docker

Se puede pensar que el uso de contenedores puede añadir cierta sobrecarga a la ejecución de aplicaciones, requiriendo sistemas con mayores prestaciones para no aumentar el tiempo entrenamiento de modelos e inferencia, en comparación a la ejecución nativa de las aplicaciones en el servidor. Un estudio realizado por IBM [32] muestra como el consumo de CPU, Memoria y operaciones de entrada y salida sobre el disco son similares a la forma nativa. La única sobrecarga (prácticamente inapreciable) que añade Docker está en la latencia de la red si se usa NAT en el mapeo de puerto entre el host y el contenedor. La Fig. 12 muestra la latencia de la red ante una determinada inyección de carga, comparando las versiones nativas, Docker y máquinas virtuales.

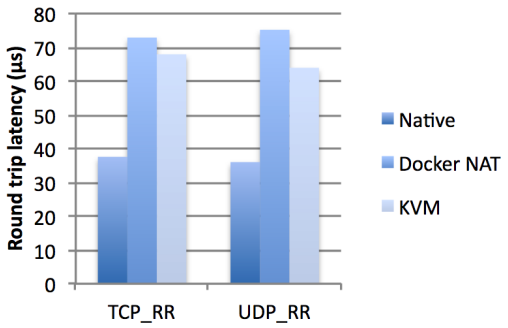


Figura 12. Latencia de red en Docker con NAT

En este proyecto, el uso Docker constituye la base sobre la que se definirán los distintos servicios a proporcionar a los Data Scientist y que finalmente, el uso de cada uno de ellos constituirá el workflow a seguir en cada desarrollo de modelos de Machine Learning. Los diferentes servicios que los contenedores de Docker Engine pretende cubrir son:

- Servicios que proporcionen un entorno de desarrollo remoto para los Data Scientist. A partir de imágenes base de R y Python, se pretende proporcionar entornos donde todas las librerías y dependencias necesarias estén ya predefinidos. Estos servicios serán definidos como microservicios de desarrollo.
- Servicios que alberguen almacenes de datos para, por un lado, volcar los conjuntos de entrenamiento que serán utilizados en el modelado y, por otro lado, alberguen los datos en vivo recabados de las aplicaciones del usuario final para realizar tareas tales como, por ejemplo, reentrenamiento de modelos. Estos servicios serán denominados microservicios de producción.
- Servicio que albergue dashboards de monitorización para que los Data Scientist puedan hacer un seguimiento de la calidad de las predicciones de los modelos.

Todas las imágenes que dan lugar a la creación contenedores serán albergadas en un servidor local de aplicaciones, concretamente en un registro interno de imágenes que la plataforma Docker proporciona, denominado Registry, y que actúa a modo de repositorio dedicado únicamente a albergar imágenes de Docker.

5.1.1.2.- Docker Swarm

Otro de los productos que Docker ofrece es el denominado Swarm. Docker Swarm permite desplegar un pool de contenedores Docker en múltiples nodos denominados workers. Estos son gestionados por otro tipo de nodos denominados manager se encargan del despliegue y mantenimiento del cluster, enviando a los workers las tareas que deben de ejecutar.

Un servicio consiste en un conjunto de tareas que pueden ser ejecutadas en los nodos de Swarm, los cuales pueden ser definidos a través de ficheros YAML. Estos servicios pueden ser replicados para ejecutarse en múltiples workers, lo que añade escalabilidad a la solución.

La Fig. 13 representa un esquema gráfico de las partes que constituyen Docker Swarm.

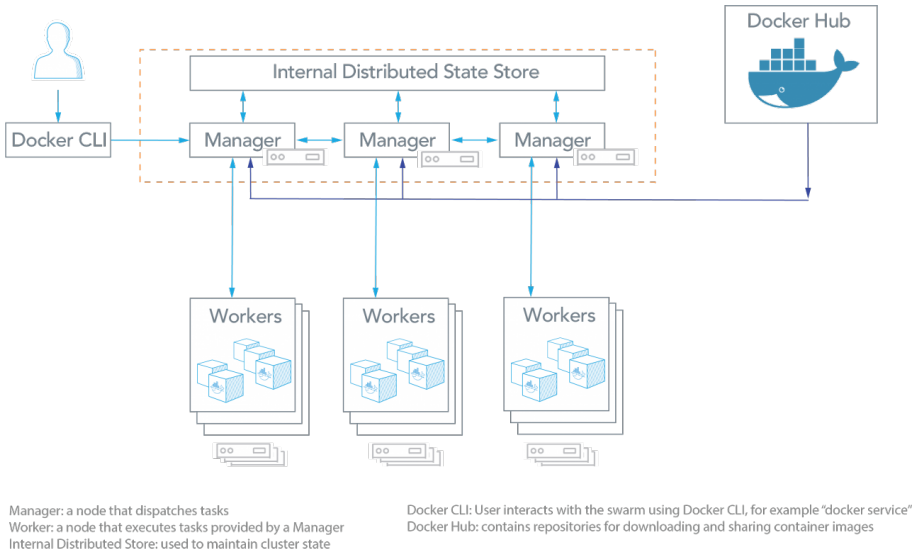


Figura 13. Esquema Docker Swarm

Si se requiere escalar los procesos de inferencia de los modelos de datos, replicando el mismo servicio en múltiples procesos, Docker Swarm daría solución a dichas necesidades.

Sin embargo, Swarm no es la única tecnología empleada para la puesta en producción de contenedores. Existe otra plataforma más avanzada y compleja que permite automatizar el workflow de puesta en producción y escalado automático de nodos llegando incluso a soportar miles de procesos en producción; esta plataforma se denomina Kubernetes.

Para finalizar con este punto, es necesario explicar las diferencias entre ambas opciones, ventajas y desventajas de cada una, y plantear Kubernetes como una posible opción para una arquitectura más avanza que Swarm en el caso de que el número de modelos en producción excesivamente elevados.

Kubernates es plataforma Open-source que proporciona un entorno para la gestión centralizada de cargas de contenedores y servicios, facilitando su configuración, automatización y crecimiento. Fue desarrollada por Google según la experiencia que estos tenían acerca de la puesta en producción de contenedores. Como se puede observar en la Fig. 14, los componentes que forman la arquitectura de Kubernetes son más numerosos y complejos, requiriendo una mayor infraestructura para poder desplegar todos los componentes.

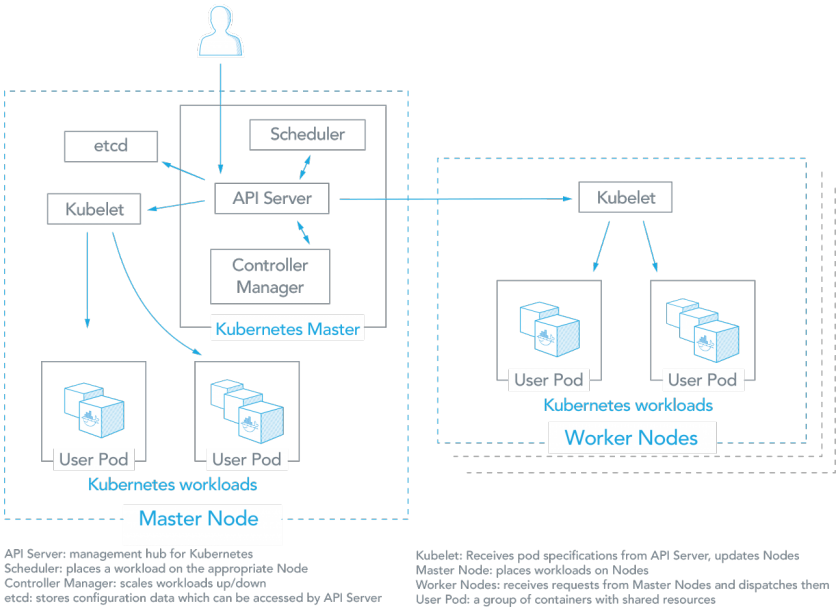


Figura 14. Arquitectura Kubernetes

Podemos resumir que los puntos más notables en los que se diferencian Kubernetes y Swarm son:

	Kubernetes	**Swarm**
Definición de la aplicación	A través de *Pods*, formando grupos de contenedores que comparten recursos.	Como servicios en un cluster Swarm.
Escalabilidad	A través de la réplicas de los servicios en *Pods*. Pueden replicarse manualmente o automáticamente.	A través de archivos YAML usando Docker Compose. Las réplicas deben definirse manualmente.
Alta disponibilidad	Replicando los *Pods* a lo largo de los nodos de Kubernates.	Replicando los servicios a lo largo de los nodos de Swarm.
Balanceo de carga	Los Pods son expuestos a través de un servicio que es usado como balanceador de carga.	Swarm proporciona un DNS que puede ser usado para distribuir las peticiones a u servicio.
Auto-escalado de aplicaciones	Disponible automáticamente replicando el número de Pods, En base a la utilización de CPU.	No disponible directamente. Para cada servicio se tiene que especificar el número de tareas a ejecutar.
Health Checks	Es capaz de comprobar si las aplicaciones se encuentran vivas (responden las peticiones) o están en modo lectura (no ocupadas)	Únicamente a nivel de contenedor. Revisando que estos se encuentran en estado de ejecución.
Almacenamiento	Existen dos APIs para el almacenamiento: una provee acceso a backends de almacenamiento y la segunda provee abstracción	Soporte mediante volúmenes montados en los contenedores.
Descubrimiento del servicio	A través de un servidor DNS disponible como addon.	A través de los nodos manager. Los cuales asignan a cada servicio un nombre DNS
Rendimiento y escalabilidad	Ha sido escalado hasta 5000 nodos.	Ha sido escalado hasta 30000 contenedores y 1000 nodos con un manager de Swarm.

Tabla 2. Comparativa de plataformas de producción avanzadas

5.1.2.- Interfaces para el desarrollo remoto

Uno de los servicios que se pretende dotar a la arquitectura es el de permitir a los desarrolladores de modelos la ejecución remota del proceso de análisis y entrenamiento de los modelos. La fase de entrenamiento puede requerir mucho tiempo y una gran capacidad de cómputo qué los desarrolladores no poseen en sus equipos locales. Por este motivo, se propondrán dos tecnologías que permitirán la ejecución remota de código de los lenguajes R y Python como en, por ejemplo, un servidor dedicado.

5.1.2.1.- RStudio Server

Plataforma Open-source que provee una interfaz web a una versión de R ejecutándose en un servidor remoto Linux. El IDE es similar a RStudio lo que facilitaría la adaptación del usuario al entorno de trabajo ya que actualmente es el IDE que el equipo de desarrollo está utilizando; incorpora las herramientas de visualización de gráficos, historial, debugging y gestión del workspace.

El uso de RStudio Server implica numerosos beneficios para el caso de estudio:

- La posibilidad de acceder al workspace desde cualquier equipo y localización.
- Facilita compartir el código, datos y otro tipo de ficheros con el resto de miembros del equipo.
- Permite a múltiples usuarios acceder a mejores recursos de computación disponibles en un servidor dedicado.

El despliegue de RStudio Server se realizará a partir de una imagen base predefinida que contenga las librerías y dependencias específicas que el equipo considere oportunas, de tal manera que cada desarrollo de los distintos modelos R se basen en las mismas librerías con el mismo número de versión. Una vez se ejecute la imagen el contenedor se creé, los Data Scientist podrán tener acceso remoto a su directorio de trabajo y mayor capacidad de cómputo.

La versión Open-source de RStudio Server es gratuita. Existe otra versión denominada Enterprise que proporciona un mayor número de características como: Múltiples sesiones de R, balanceo de carga, dashboard administrativo, autenticación de usuarios, etc… En el caso de esta propuesta, el uso de RStudio Server no será centralizado, sino que habrá tantos procesos como contenedores se ejecuten por usuario.

5.1.2.2.- Pycharm

Es un IDE de desarrollo para Python. A diferencia de RStudio Server, que proporcionaba un servidor remoto donde ejecutar, almacenar y gestionar los ficheros de R, Pycharm únicamente proporciona la ejecución remota de código, a través de conexión ssh con el servidor o haciendo uso de contenedores. El usuario debe de gestionar localmente sus scripts. A la hora de ejecutar código remoto Pycharm proporciona la opción de desplegar un contenedor, ejecutar el código y remotamente y retornar el resultado al usuario cuando el contenedor finalice su ejecución. Por cada ejecución se crea un nuevo contenedor el cual monta el directorio del proyecto de Pycharm al contenedor para su ejecución; posteriormente el contenedor es eliminado. Todas las dependencias y librerías deben estar instaladas en el contenedor que se va a utilizar como intérprete remoto.

Para permitir que Pycharm se conecte al daemon de Docker en el servidor se debe de exponer un puerto de escucha asociado al daemon. Para ello, se debe modificar el archivo donde se define el servicio docker:

```
sudo nano /lib/systemd/system/docker.service
```

Añadir IP y puerto donde el daemon será accesible:

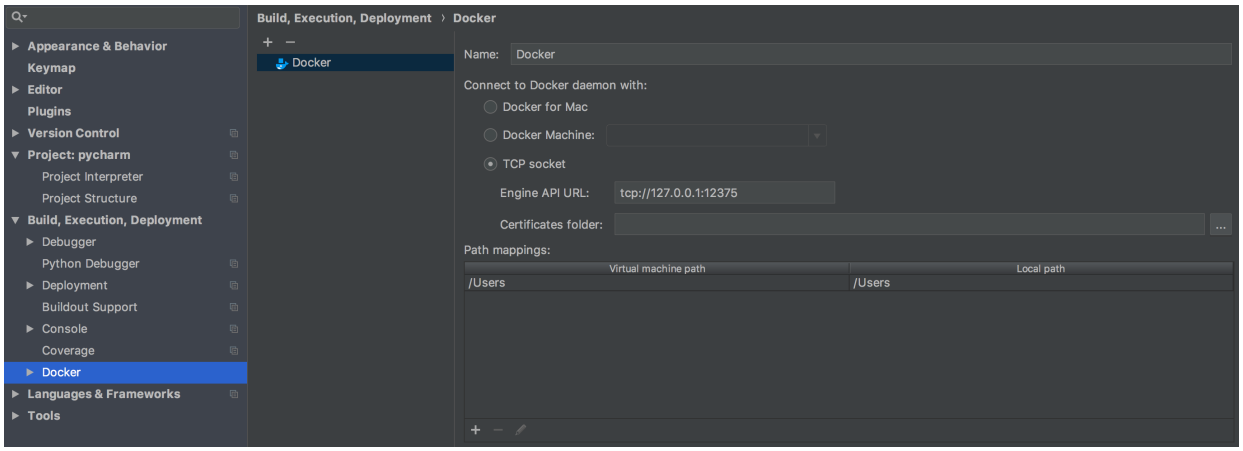
```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375
```

Reiniciar la configuración y servicio para aplicar los cambios:

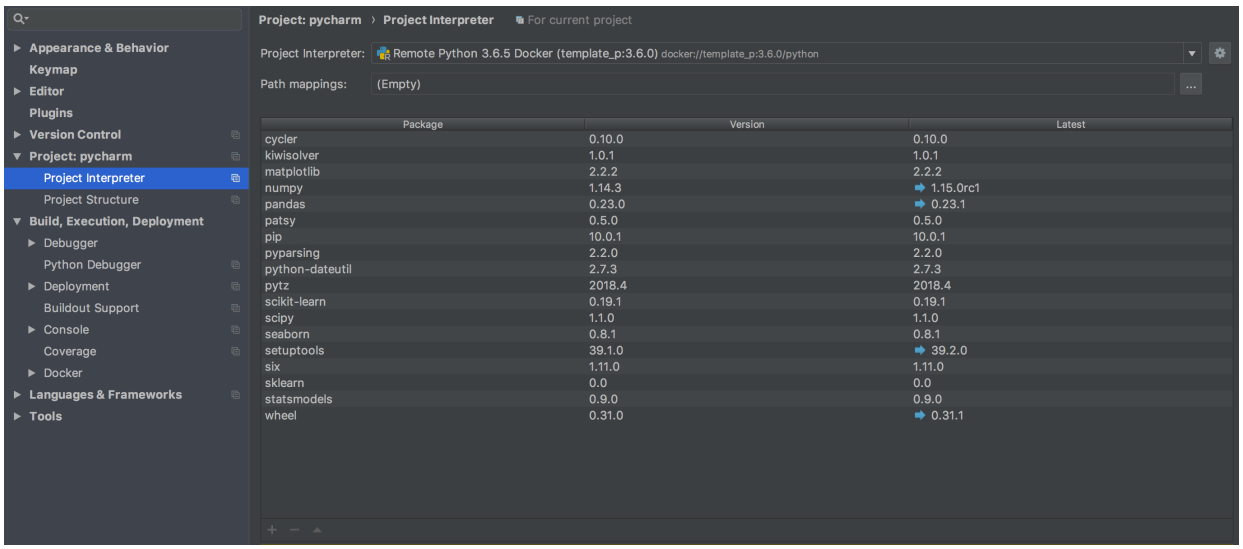
```
systemctl daemon-reload
```

```
sudo service docker restart
```

A continuación desde Pycharm ya podemos agregar el nuevo contenedor a nuestra lista de intérpretes a través un socket TCP:



Seleccionamos la imagen de Python y a partir de este instante todas las ejecuciones se realizarán remotamente.



5.1.3.- Cassandra

Cassandra es el sistema de gestión líder de bases de datos distribuidas NoSQL y orientado por columnas, diseñada para gestionar grandes cantidades de datos. Las estructuras de datos usadas en Cassandra son más específicas que las utilizadas en bases de datos relacionales, permitiendo mayor velocidad de lectura y escritura; un uso típico de Cassandra es en aplicaciones web que trabajan con peticiones en tiempo real.

Las características más relevantes que Cassandra aporta son:

- Arquitectura masivamente escalable.
- Arquitectura descentralizada. Los datos pueden ser escritos y leídos desde cualquier nodo.
- Escalado lineal. Cuantos más nodos se añadan el rendimiento de Cassandra aumentará.
- No hay un único punto de fallo. Los datos están replicados en diferentes nodos.
- Detección de fallos y recuperación automática.
- Modelado de datos dinámico. Soporta diferentes tipos de datos que agilizan las lecturas y escrituras.
- Protección de datos. Se proporcionan mecanismos para realizar backups y restauraciones.
- Consistencia de datos a lo largo de toda la arquitectura distribuida.
- Compresión de los datos. Los datos pueden ser comprimidos hasta un 80% sin ninguna sobrecarga.
- CQL (Cassandra Query Language) es posible realizar operaciones frente a Cassandra utilizando la misma lógica que con BD´s SQL, en el sentido de operar con tablas que contienen filas y columnas

En cuanto al motor de almacenamiento de Cassandra, este cuenta con 3 elementos fundamentales:

- CommitLog: Almacenan los logs de todas las operaciones realizadas sobre cada nodo de Cassandra. Toda operación realizada sobre Cassandra será escrita en un commit log antes de ser escrita a una Memtable. Este elemento permite, en caso de apagado brusco del sistema, aplicar las operaciones no realizadas sobre las Memtables una vez se ha reiniciado el sistema. Además, la escritura en disco de los commit logs se realiza de manera eficiente, almacenando en segmentos la información a escribir en el CommitLog y una vez estos llegan a su límite de capacidad se procede a su escritura en disco.
- Memtables: Son estructuras de datos almacenadas en memoria las cuales Cassandra utiliza como buffers de escritura. Existe una memtable por tabla. Posteriormente estas son volcadas a disco y pasarían a ser SSTables. Las dos opciones que se pueden dar para ocurra el volcado a disco son:

    - El uso de memoria de la memtable excede el umbral fijado por el administrador.
    - El Commitlog sobrepasa su tamaño máximo y fuerza a las memtable a volcar su contenido en disco para liberar a los segmentos del commitlog.
- SSTables: Son los ficheros de datos inmutables de Cassandra que usa para mantener persistentes los datos en disco. Los datos son volcados en las sstables desde las memtables o son recibidos de otros nodos de Cassandra, ya que esta trata de compactar múltiples tablas SSTables en una sola. Cada tabla SSTable a su vez esta formada por un múltiples componentes almacenados en diferentes ficheros.

A la hora de operar con Cassandra es importante definir los siguientes conceptos:

- Keyspace: Define un grupo de columnfamilies en Cassandra. El concepto es similar a la creación de bases de datos en los lenguajes relacionales, las cuales albergan a un conjunto de tablas.

- Columnfamily: Es similar al concepto de tabla en bases de datos relacionales. El concepto de columnfamily agrupa a una colección de filas. Cada fila contiene una colección de columnas ordenadas, por lo que dan estructura a los datos del usuario.
- Partition Key: Es el identificador único de cada fila en Cassandra. Determina en que nodo se alberga los datos y se encarga de su distribución a través de los nodos.
- Clustering Key: Define un proceso de almacenamiento el cual ordena los datos en la partición a través de su valor. Todas las consultas realizadas que requieran extraer información en un determinado orden debe definir o acceder a las Clustering key correspondientes.
- Primary Key: Se compone de la Partition Key más las Clustering Key que se hallan definido.

A continuación se muestra una imagen de la estructura de una base de datos orientada por columnas:

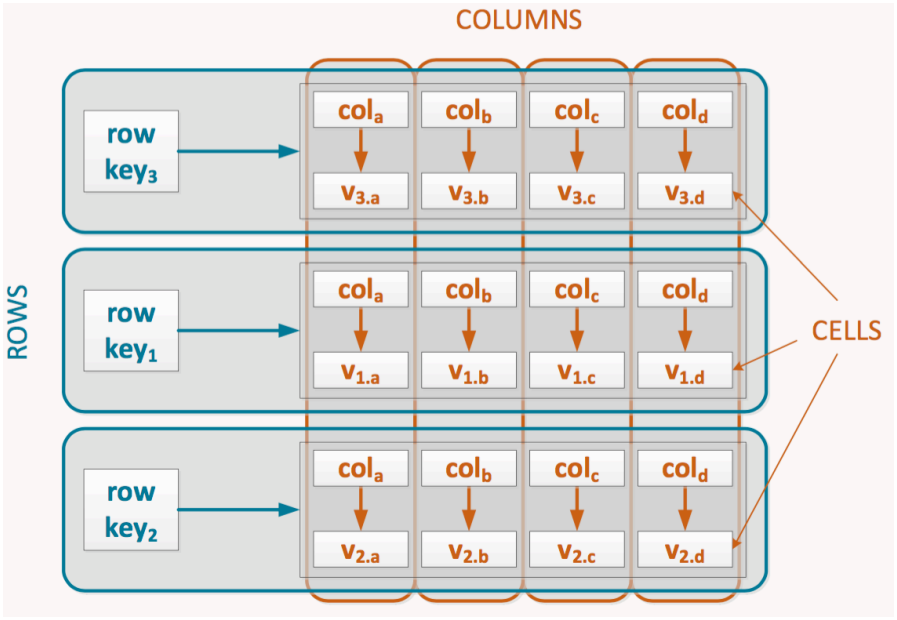


Figura 15. Esctructura de Cassandra

Cassandra será utilizada en esta arquitectura para almacenar los datos históricos, nuevos datos que se obtengan de las apps de usuario y predicciones realizadas por los modelos. Debido a la diversidad de datos que se puedan recibir en el futuro, se ha optado por una base de datos NoSQL para almacenar datos no estructurados.

5.1.4.- A/B Testing

Hasta este punto se han presentado un conjunto de alternativas tecnológicas que pretenden cubrir diferentes puntos del diseño de la arquitectura. Sin embargo, un componente no tecnológico que esta propuesta pretende cubrir es el testing de modelos ante nuevos datos, denominados live data, enviados para realizar inferencia. Por testing de modelos no estamos haciendo referencia a técnicas que permitan hacer pruebas sobre código o funciones de estos, sino, que ante un conjunto de modelos entrenados para resolver el mismo problema, se pretende estudiar que formas existen para medir el rendimiento de cada uno ellos de cara a saber que configuración es la más adecuada.

Para ilustrar el problema que se pretende resolver, se expone el siguiente ejemplo. Un proyecto relacionado con la ciencia de datos pretende desarrollar un modelo de Machine Learning para resolver un determinado problema, para el cual participan 2 Data Scientist. Cada uno realiza por separado tanto la parte de análisis de datos como de diseño y desarrollo, dando lugar a diferentes conclusiones en cada uno de los estudios realizados. Como resultado, por lo tanto, se obtienen 2 modelos diferentes o un mismo modelo con diferentes configuraciones de hiperparámetros. Si las métricas de evaluación de ambos modelos los consideran aptos para su implementación en producción ¿Cuál de ellos se ha de implementar?.

Ante la pregunta del ejemplo planteado se pueden a portar diferentes soluciones.

- El modelo que mejores resultados haya obtenido ante los datos de validación.
- Llevar ambos modelos a producción, monitorizar sus resultados durante un cierto período de tiempo y volver a evaluar la calidad de ambos finalizada la ventana temporal fijada.
- Llevar ambos modelos a producción, y de manera estocástica retornar al usuario las predicciones de un modelo o de otro.

En este apartado se pretende dar una solución sobre que técnicas aplicar para resolver las cuestiones planteadas. Primeramente nos centraremos en la técnica A/B Testing y se finalizará proponiendo una técnica estocástica más avanzada denominada Multiarmed Bandit.

La técnica de A/B Testing nace del mundo del marketing digital y analítica web, con el objetivo de evaluar 2 o más versiones de una misma página web idénticas salvo por una variación que puede afectar al comportamiento del usuario y a la predisposición de este a realizar determinadas operaciones sobre la web. El proceso se resume en: Desplegar ambas versiones de la misma web, redirigir aleatoriamente a los usuarios entrantes a una u otra versión (Grupo A y Grupo B), esta división en grupos se mantiene activa hasta que el desarrollador web tome la decisión (p.e. haciendo uso de Google Analytics o haciendo uso de test estadísticos) de que la versión A es mejor que la B o viceversa. Finalmente, todos los usuarios son redirigidos a la versión que ha tenido éxito y la otra versión es descartada.

El concepto de A/B testing se puede aplicar de manera similar a la evaluación de modelos de Machine Learning que se quieren desplegar en producción y ver que comportamientos se obtienen ante las peticiones de usuario. En este caso, en vez de tener dos versiones A y B de una web: “control” (la web principal) y “variation” (la versión modificada), se pueden tener 2 o más modelos. El modelo A es el que definiremos como “control” o “champion”, es el modelo que actualmente se encuentra en producción, ya sea por que ha sido el que mejor puntuación ha obtenido en la fase de validación o por que no se han realizado más configuraciones del mismo; es sobre

el que la aplicación de usuario realiza la inferencia, almacena los resultados en la base de datos y devuelve las predicciones a esta. Por otro lado, el modelo B, será el modelo "variation" o "challenger", este modelo no se comunica con la aplicación de usuario, únicamente recibe los datos y almacena los resultados de la inferencia en la base de datos con el objetivo de ser comparados posteriormente con la versión A del modelo.

Una diferencia notable del enfoque de A/B Testing aplicado a modelos de Machine Learning con respecto al uso, por ejemplo, en aplicaciones web es que en el último caso se tiene que dividir el tráfico en un 50% para la versión A y otro 50% para la versión B, puesto que dos usuarios no pueden estar navegando por 2 webs simultáneamente. En el caso del uso en Machine Learning un único flujo de datos no se tiene por que dividir, se pueden evaluar en paralelo por los *n* modelos que se encuentren en producción.

Se pueden dar numerosas situaciones por las que se quiera sustituir el modelo A por el B.

- La precisión de A ha decaído de un cierto umbral debido a cambios en la distribución de entrada. El modelo B tiene la misma configuración que A pero ha sido reentrenado con nuevos datos recabados, adaptándose mejor a la distribución de los nuevos datos y, por lo tanto, mejorando la precisión de A. Existen dos técnicas para abordar el proceso de reentrenamiento:
    - Sliding window. Consiste en fijar una ventana temporal la cual mantiene su tamaño a largo del tiempo. A la hora de realizar el reentrenamiento, se cogen los datos desde el último dato hasta el tamaño fijado; el resto son descartados.

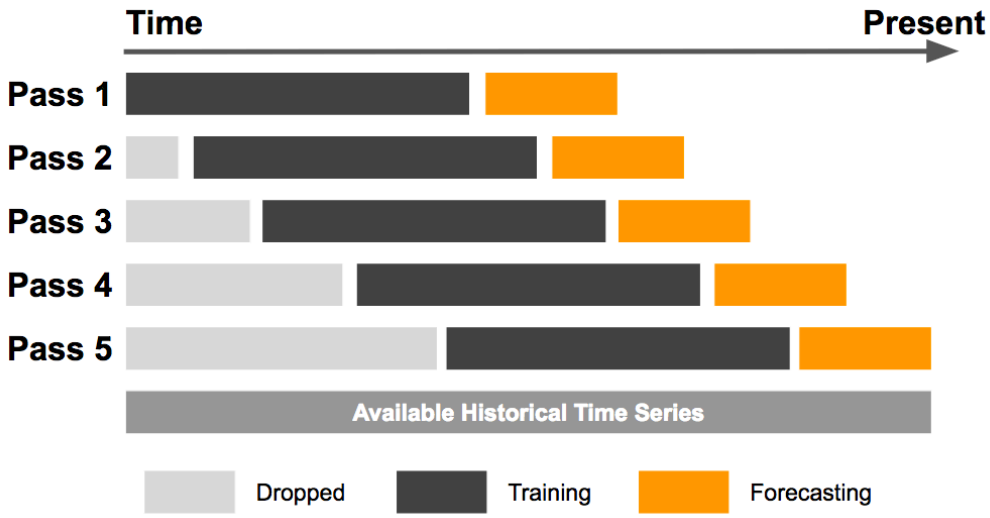


Figura 16. Sliding window

- Expanding window. A diferencia de la técnica anterior, no se descarta ninguno de los datos. El reentrenamiento se realiza con los datos disponibles hasta la fecha.

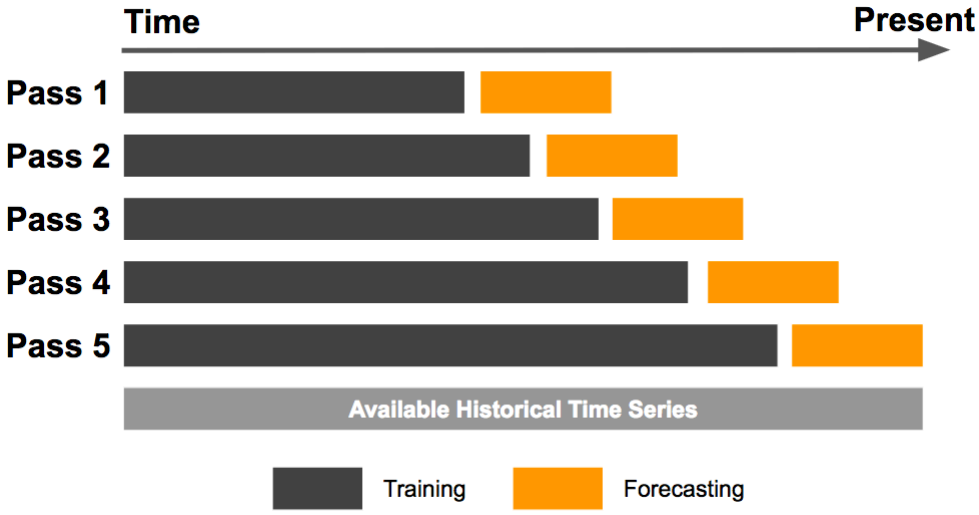


Figura 17. Expanding window

- Otra situación común que se puede dar es que el modelo B sea un modelo diferente a A, bien sea por que ha sido entrenado con una configuración de hiperparámetros diferente o por que la técnica de modelado sea diferente. Si se da la situación de que la función *loss* (función que mide la diferencia entre los datos predichos y los reales) del modelo B supera a la del modelo A, también podría ser deseable realizar el cambio y el modelo B pasaría a ser la versión "champion*"* y el A el "challenger*"*.

El criterio para la sustitución de un modelo "challenger*"* por un "champion*"* se puede realizar en base a: métricas de monitorización o mediante test estadísticos.

El primer criterio y más sencillo de llevar a cabo consiste en aprovechar el dashboard visualización sobre las métricas de visualización que se pretende implementar junto a la arquitectura en este proyecto. Como se explicará más adelante, esta aplicación web permitirá a los Data Scientist llevar un control de la calidad de los modelos en producción en base a distintas métricas de medición. Por lo tanto, en base a estas métricas y según se crea oportuno se podrá realizar el cambio de "challenge*r"* a "champion*"*.

El segundo criterio consiste en realizar test estadísticos sobre las métricas del modelo A y B, nos permite obtener cuando la diferencia de medias es suficientemente significativa para realizar el cambio de "challenger" a "champion". Para ello se establece como hipótesis nula que el promedio del valor de la métrica de rendimiento no cambie) y la hipótesis alternativa que el promedio del valor de la métrica de rendimiento cambie, se calcula el p-valor y se toma la decisión.

En resumen, los pasos a seguir para implementar A/B Testing mediante el segundo criterio serían:

1. Dividir los modelos en un grupo de control (A) y experimentación (B).
2. Observar las métricas de rendimiento de los modelos en ambos grupos.

3. Realizar test estadístico sobre los valores para si existen diferencias significativas para realizar un aviso. Tests como t-test y chi-square pueden ser empleados sobre las medias de las métricas de cada modelo.

5.1.5.- Multiarmed-bandit

Los algoritmos Multiarmed-bandit [33] son una técnica estocástica más compleja que A/B Testing, la cual permite evaluar diferentes modelos ante nuevos datos de entrada. Surge del mundo de la analítica web, por lo que el concepto que se pretende exponer será una adaptación de los algoritmos Multiarmed a los modelos de aprendizaje. Su implementación se puede llevar a cabo a través diversos algoritmos, de los que son más conocidos: Epsilon-Greedy, Softmax Algorithm, The Upper Confidence Bound Algorithm (UCB).

Las conceptos que forman parte de todo algoritmo Multiarmed-bandit son:

- Arm: Hace referencia a una opción disponible que se puede explorar. Es el equivalente al concepto de alternativas A, B, etc… de A/B Testing. En nuestro caso, un "arm" se corresponderá con un modelo de aprendizaje.
- Reward: Medida cuantitativa que indica el éxito de un modelo. Cada "arm" tiene su reward acumulado a través de las diferentes iteraciones del algoritmo.
- Bandit: Colección de "arms". Es decir, conjunto de modelos que se quieren evaluar, a excepción del que mejor "reward" tenga (Best arm).
- Exploración: Fase de los algoritmos Multiarmed en la que se evalúan los "arms" que no tienen el máximo "reward" acumulado.
- Explotación: Se utiliza el mejor "arm" (mayor valor de reward). Es equivalente a realizar una predicción de nuevos datos con el modelo que mejor valor de "reward" tenga hasta el momento.

Para ilustrar un ejemplo de la aplicación de esta técnica se usará el algoritmo Epsilon-Greedy. En la Fig. 18 se presenta un diagrama de los componentes Epsilon-Greedy.

Supongamos que se quieren evaluar *N* + 1 modelos diferentes entre sí, ya sea por que se utilizan diferentes técnicas o configuraciones que resuelven el mismo el problema. Uno de ellos presenta las mejores métricas de evaluación en la fase de entrenamiento y será seleccionado como "Best arm" comenzando con el "reward" más alto. Supongamos que la calidad de los resultados de la evaluación en la fase de entrenamiento de los *N* restantes es inferior a la de "Best arm", pero se quiere probar su rendimiento ante live data. Estos *N* modelos constituirán el "bandit*"*. El último paso para dar comienzo la ejecución del algoritmo sería fijar el valor *epsilon*, que indica la frecuencia con la que se debería explorar uno de los "arms" disponibles; típicamente se fija el valor de *epsilon* en 0.2.

En resumen, los pasos de Epsilon-Greedy son:

1. Se genera un número aleatorio, a partir de una distribución de probabilidad uniforme, en el intervalo [0, 1] y se comprueba si este es valor es mayor que *epsilon*; en ese caso comenzará la fase de explotación y se devolverá una predicción con el mejor "arm" y volveremos al comienzo de la iteración, sino comenzará la fase de exploración de los *N* "arms".
2. Si el valor aleatorio generado es menor o igual que *epsilon* comienza la fase de exploración. En esta se vuelve a generar un número aleatorio que puede tomar los valores [1, *N*] uniformemente. Con el valor obtenido se selecciona el "arm*"* y se realiza la predicción con el modelo correspondiente.
3. Se recalcula el "reward*"* acumulado de ese brazo y se compara como "Best arm*"*. De tal forma que si el nuevo "reward" supera al máximo, "best arm" es sustituido por el nuevo modelo.
4. Finaliza la iteración.

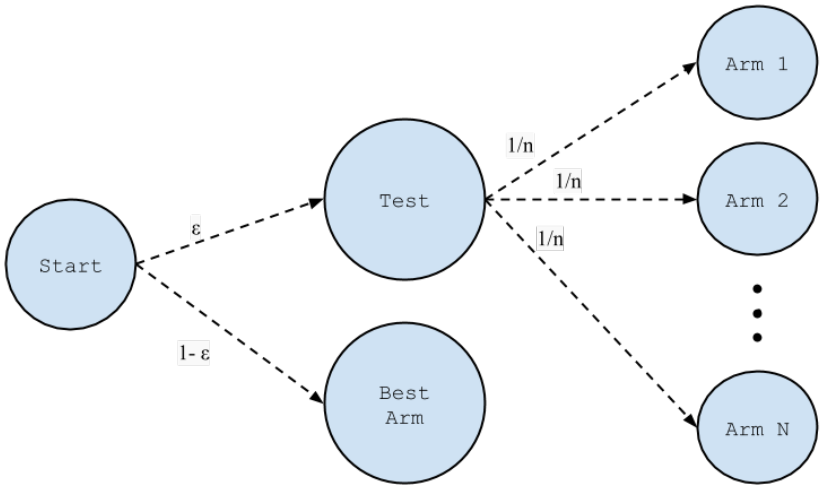


Figura 18. Multiarmed-bandit Epsilon-Greedy

La técnica de Multiarmed-bandit presenta soluciones para ambos inconvenientes, por un lado exploran un mayor número de "challengers" realizando un mayor recorrido en las transiciones y solo gastan sus recursos en las mejores opciones en vez de gastar tiempo en opciones inferiores que son sobre-exploradas durante un típico A/B Testing. Gradualmente Multiarmed- bandit ajustan las mejores opciones a lo largo del tiempo.

Como comparativa entre los métodos A/B Testing y Multiarmed-Bandit podemos diferenciar:

- La técnica Multiarmed no gasta tantos recursos como A/B Testing. Puesto que el "bandit" de opciones alternativas solo se explora con una probabilidad *epsilon*. En A/B Testing cada live data recibido es predicho por el "champion" y todos los "challengers".
- Gradualmente Multiarmed ajusta la mejor opción automáticamente a lo largo del tiempo. A/B Testing requiere tomar decisión por parte del usuario.
- Permite comprobar la satisfacción del usuario ante inferencias de modelos que no sea el "champion". Por ejemplo, se pueden dar resultados de opciones de

modelos más conservadores o más agresivos con una probabilidad *epsilon* para determinado tipo de problemas.

La opción de Multiarmed-bandit se expone como técnica avanzada. En la propuesta de la arquitectura se implementará la técnica de evaluación de A/B Testing mediante un dashboard de monitorización.

5.1.6.- Dashboard de monitorización

Como último componente del sistema de producción que se ha estado describiendo, se pretende aportar una aplicación que permita la monitorización y actualización de los modelos de Machine Learning que se encuentren en producción.

Para cubrir este apartado se propone la creación de un dashboard servido a través de una aplicación web. Dicha aplicación permitirá a los Data Scientist realizar controles de la calidad acerca de los modelos desplegados que se encuentren realizando inferencia a partir de live data y modelos que se encuentre en fase de testing mediante la técnica A/B Testing.

En primer lugar se presenta un pequeño resumen de las tecnologías disponibles en el mercado que permiten facilitar la creación y gestión de dashboards interactivos. Entre las que se encuentran:

5.1.6.1.- Tableau

Es una empresa dedicada al desarrollo de productos de visualización de datos de manera interactiva. Ofrecen diversos productos, los cuales son mas convenientes en unos casos u otros dependiendo de las necesidades de usuario. Cada uno permite al usuario desarrollar dashboards interactivos a través de una interfaz web y construir gráficos a partir de diversas fuentes de datos como bases de datos relacionales, bases de datos en la nube, hojas de cálculo, etc...

Los productos que ofrece Tableau son:

- Tableau Desktop: Es una herramienta enfocado al desarrollo de dashboards. A través de una aplicación de escritorio se permite la creación de elementos a través de una interfaz de usuario y conexión de estos a diferentes fuentes de datos; no se requieren conocimientos de programación.
- Tableau Server: Permite la creación de un servidor dedicado a servir dashboards desarrollados con Tableau Desktop dentro de la organización. Los usuarios pueden interactuar con los mismos desde múltiples dispositivos.
- Tableau Online: Misma funcionalidad que Tableau Server pero con alojamiento en la nube. Se suele emplear como alojamiento para mostrar la información a gente externa a la empresa. Es comúnmente utilizado para mostrar resultados a los clientes.
- Tableau Reader: Aplicación sin coste alguno que permite abrir y visualizar dashboards construidos con Tableau Desktop.

El coste de la obtención de las licencias Tableau Desktop y Server para cubrir tendría un coste de 70$ al mes por usuario.

La opción de usar Tableau como software para el desarrollo una herramienta de monitorización se descarta debido al coste y dependencias entre los productos que ofrece Tableau para crear y compartir las visualizaciones.

5.1.6.2.- SAP Lumira

El propietario de esta tecnología es SAP. Permite la creación de dashboards utilizando como fuentes de datos archivos .xlsx, .csv, bases de datos relacionales y Big Data, además de integrarse con los datos almacenados en SAP HANA. SAP Lumira permite preparar datos y posteriormente construir gráficos interactivos a partir de estos los cuales se pueden publicar en otras herramientas de SAP.

Los costes de las licencias de SAP asciende hasta los 1000$ por usuario al mes.

5.1.6.3.- Power BI

Herramienta comercial de Microsoft, la cual ofrece un conjunto de aplicaciones de análisis de negocios permitiendo analizar datos y compartir información. La información mostrada por Power BI se obtiene a partir de un conjunto de orígenes de datos y se encuentra actualizada en tiempo real en todos los dispositivos. Los productos más destacados que se ofrecen son:

- Power BI Mobile: Permite acceder a los datos e informes creados desde cualquier dispositivo móvil.
- Power BI Desktop: Es la herramienta principal de Power BI. Implementa las opciones de creaciones de visualizaciones de datos e informes.

El inconveniente de Power BI es que no soporta Cassandra como fuente de origen de los datos.

Las herramientas comerciales vistas presentan un coste a excepción de Power BI Desktop. Además puesto que el uso que se le pretende dar al dashboard, a parte del análisis de métricas de monitorización, es que posibilite actualizar modelos automáticamente, además de posibilitar la conexión a bases de datos NoSQL. Ninguna de estas opciones comerciales implementa esta característica, ya que están más orientadas a la creación de informes y comunicación de resultados hacia usuarios sin un perfil técnico. Por lo que se optará por el desarrollo de un dashboard personalizado.

La opción más común es utilizar un conjunto de tecnologías web para crear un dashboard totalmente personalizado. Utilizando elementos HTML y estilos CSS para construir el dashboard, javascript para crear la lógica y dinamismo y D3.js como librería para realizar los gráficos de visualización.

El problema de esta solución es que se requieren conocimientos de desarrollo web de frontend para realizar una interfaz de usuario interactiva de calidad. La parte del backend (acceso a los datos y lógica del cálculo de las métricas) sería necesaria definirla también puesto que existen numerosas tecnologías para dicho fin.

En definitiva, esta solución es la que más flexibilidad aporta al diseño del dashboard y no tendría coste tecnológico alguno. Sin embargo, el tiempo de desarrollo requerido es mayor y se requiere personal técnico para el desarrollo frontend. El Data Scientist no podría llevar a cabo esta tarea desarrollo y tendría que colaborar con un desarrollador web, lo que implica un mayor tiempo de desarrollo debido al tiempo que este debe emplear para el correcto diseño de las visualizaciones de las métricas de rendimiento.

Para intentar solucionar este problema, existen frameworks en lenguajes como R y Python que permiten la construcción de aplicaciones web interactivas para la visualización de datos. De tal manera que el desarrollador no requiere de conocimientos avanzados de tecnologías web y puede desarrollar fácilmente este tipo de aplicaciones.

En el área de la ciencia de datos los frameworks más utilizados para este tipo de tareas son: Shiny para R y Dash para Python, ambas Open-source y sin coste alguno.

5.1.6.4.- Shiny

Shiny es un paquete de R que permite construir aplicaciones web interactivas directamente desde el propio lenguaje R. Permite crear rápidamente sencillos dashboards abstrayendo al desarrollador de elementos HTML, Javascript o CSS, aunque estos también pueden ser incorporados si se desean.

Para facilitar el despliegue y gestión de las aplicaciones web desarrolladas se proporcionan dos alternativas dentro de la tecnología Shiny:

- Uso de Shiny server (Open-source) para despliegue local. Con opciones free y pro. La diferencia entre una y otra versión es que la versión pro implementa mecanismos de seguridad, autenticación y control de usuarios a las aplicaciones, permite escalar aplicaciones a través de multiprocesos de R, y ofrece conectores a determinadas bases de datos particulares.
- Uso de Shiny.io para despliegue en la nube. Con opciones free y pro. La versión free únicamente permite albergar 5 aplicaciones como máximo con una actividad de 25 horas máximo al mes (si se excede este valor la aplicación deja de estar disponible). A partir de aquí, si se desean mayores valores de los descritos se pasaría a contratar diferentes modelos de pago para obtener mayores prestaciones.

5.1.6.5.- Dash

Es un framework de Python para construir aplicaciones web de análisis de datos (misma función que Shiny pero en Python) que implementa MVC(Modelo, Vista, Controlador). Está desarrollado sobre Plotly.js, React y Flask. Las ventajas de Dash son:

- Sencillez a la hora de construir dashboards. No requiere de conocimiento acerca de Javascript, HTML, CSS, D3.js...
- Mismo lenguaje de programación. Los cálculos acera de las métricas de rendimiento se pueden realizar utilizando Python al igual que la construcción del frontend.
- Si se quieren añadir o personalizar más elementos se puede customizar cada una de las partes que componen Dash mediante componentes HTML.

Para el despliegue de estos servicios web se puede hacer uso de Flask para servir el contenido.

La opción de implementar un dashboard utilizando la tecnología Shiny se ha descartado puesto que, habiendo una tecnología similar que se permita el desarrollo en Python, se prefiere utilizar siempre que se pueda esta antes que R por motivos de la experiencia de los desarrolladores. Además puesto que el microframework Flask se ha utilizado en otros puntos de la arquitectura (creación de API´s REST en los modelos R), reutilizar este conocimiento ahorraría tiempo de aprendizaje que requeriría adaptarse a una nueva tecnología de despliegue.

5.1.7.- Métricas de monitorización

Una vez se ha propuesto la tecnología con la que construir la herramienta de monitorización web, se fijarán que métricas de evaluación de la calidad de modelos que se emplearán.

Puesto que existen numerosos tipos de modelos Machine Learning y cada uno puede tener sus propias métricas de rendimiento o evaluación, se propondrán una serie de métricas generales referidos a modelos de tipo supervisado (aquellos en los que por cada grupo de variables de entrada se conoce la variable salida) de clasificación multiclase y regresión.

Las métricas seleccionadas para la evaluación de los modelos de clasificación son:

- Accuracy: Determina el número de clases correctamente clasificadas respecto al número total de predicciones realizadas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 19. Métrica Accuracy

- Precision: Determina la calidad de un modelo determinando cuantos de los casos predichos positivamente son en realidad positivos. Esta métrica es útil para evaluar modelos en los que el coste de predecir falsos positivos es alto.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} = \frac{TruePositive}{TotalPredictedPositive}$$

Figura 20. Métrica Precision

- Recall: Determina la calidad de un modelo determinando cuantos de los casos realmente positivos son predichos correctamente por el modelo. Esta métrica es útil para evaluar modelos en los que el coste de predecir falsos negativos es alto.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} = \frac{TruePositive}{TotalActualPositive}$$

Figura 21. Métrica Recall

- F1: Proporcionando una medida balanceada entre las medidas precision y recall. La puntuación mediante la métrica F1 alcanza su mayor valor en 1 (valores perfectos de precisión y recall) y su peor valor en 0.

$$F_1 Score = 2 * \frac{precision * recall}{precision + recall}$$

Figura 22. Métrica F1

- AUC: Mide al área bidimensional debajo de la curva ROC. Proporciona una medición agregada del rendimiento en todos los umbrales de clasificación posibles. El valor de AUC se puede interpretar como la probabilidad de que una observación aleatoria sea correctamente clasificada como positiva.

$$A = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\mathrm{FPR}'(T)\, dT$$

Figura 23. Métrica AUC

- Gini coefficient: A partir de la métrica AUC se deduce el coeficiente Gini para las medidas de clasificación. Mide la ventaja de utilizar el clasificador frente a valores generados aleatoriamente. Al igual que los valores de AUC, un valor igual a uno denota que el clasificador es perfecto, mientras que un valor 0 indica que las predicciones son puramente aleatorias.

$$Gini = 2AUC - 1$$

Figura 24. Métrica Gini

Cabe destacar que estas medidas se aplican para cada clase puesto que estamos en un problema de clasificación multiclase, se debe de calcular cada una de estas métricas por clase; este proceso se denomina *binarización*.

Las métricas seleccionadas para la evaluación de los modelos de regresión son:

- R-cuadrado: Determina la calidad del modelo para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por el modelo.

$$R^2 = \frac{\sigma_{XY}^2}{\sigma_X^2 \sigma_Y^2}$$

Figura 25. Métrica R-cuadrado

- Root Mean Square Error (RMSE): Representa la raíz cuadrada del valor del error de un valor estimado frente al valor real.

$$\sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}$$

Figura 26. Métrica RMSE

- Mean Square Error (MSE): Mide el valor promedio de los errores al cuadrado; la diferencia entre el valor predicho y el real.

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}_{\hat{\theta}}\left[(\hat{\theta} - \theta)^2\right]$$

Figura 27. Métrica MSE

- Mean Absolute Error (MAE): Determina la diferencia entre dos variables continuas, comparando los valores predichos frente los valores reales.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Figura 28. Métrica MAE

5.2.- Diagrama de diseño

Con el objetivo de ilustrar como cada una de las tecnologías vistas anteriormente se comunican entre sí y ver en qué fase del proyecto de modelado de datos tienen lugar, se ha realizado un diagrama que ilustra cada uno de los conceptos.

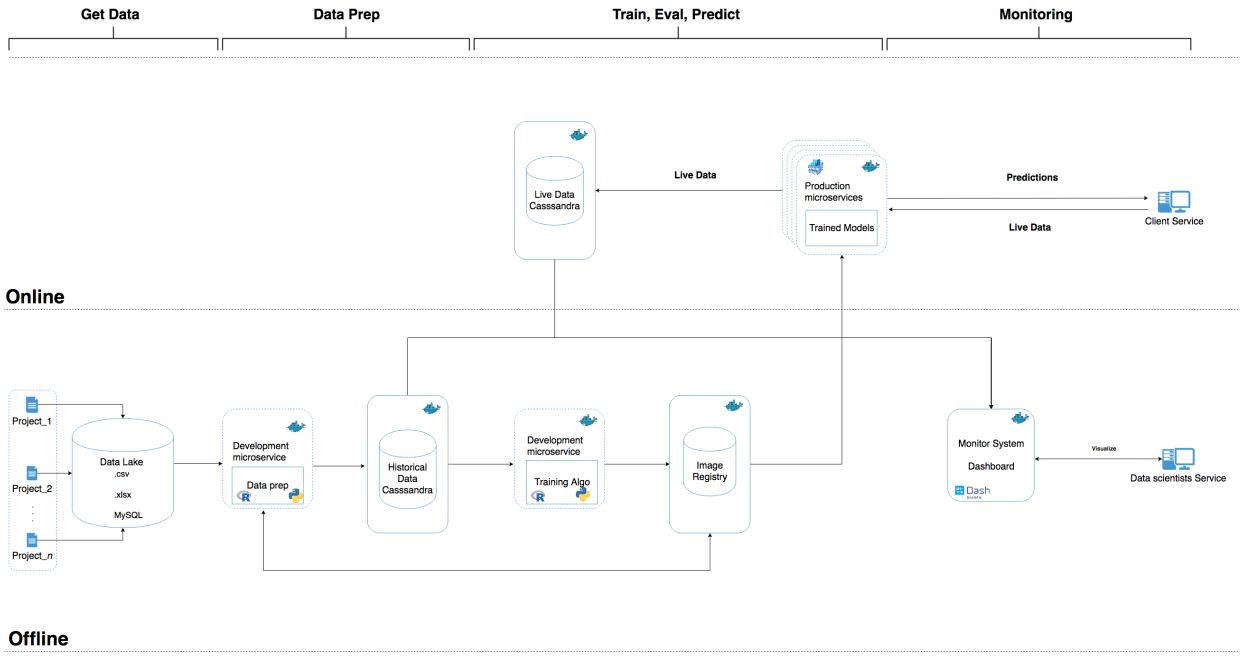


Figura 29. Arquitectura de despliegue de modelos en producción

La Fig. 29 contiene cada uno de los componentes que formarán parte del sistema. Se comienza con una base de datos en la que se dispone de todos los datos necesarios para comenzar con el proyecto, esta base de datos ya existía actualmente, como se explico en la situación actual y se denomina Datalake. Seguidamente tendríamos el proceso de preparación de datos; resumido en las Fig. 30 y 31 puesto que workflow los microservicios de desarrollo son diferentes para R y Python.

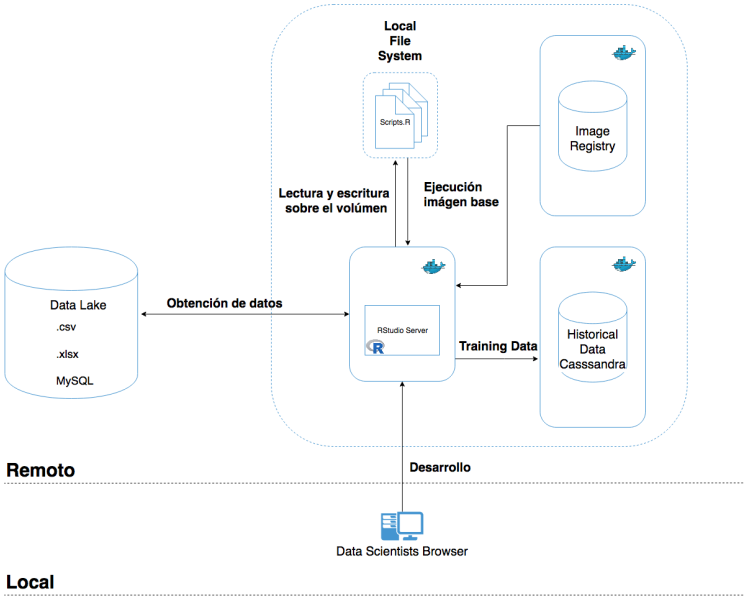


Figura 30. Preparación de datos R

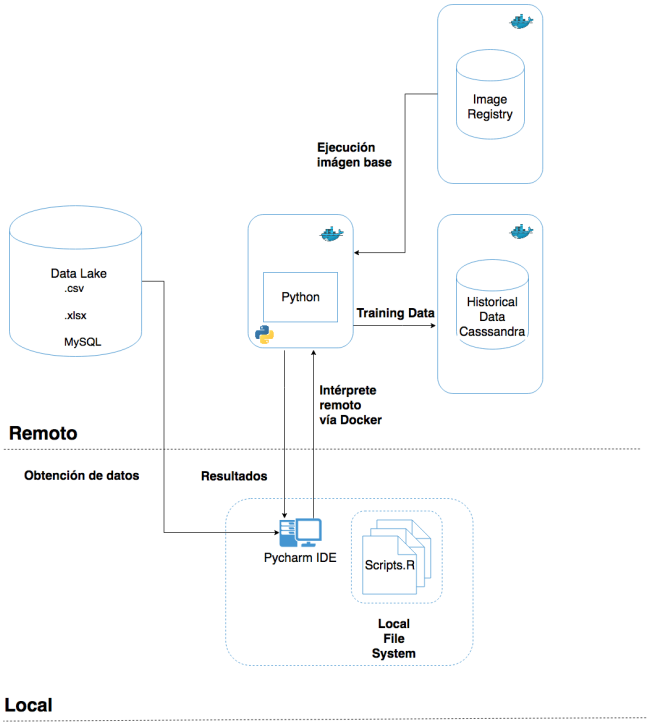


Figura 31. Preparación de datos Python

A continuación vendría el proceso de entrenamiento, el cual tiene lugar también dentro de los microservicios de desarrollo. Finalizada la elaboración de cada modelo junto con cada una de las versiones elaboradas para realizar A/B Testing se elaboran los correspondientes servicios de producción. Estos servicios de producción se pueden desplegar como instancias únicas o utilizando la tecnología Swarm en el caso de ser necesaria alta disponibilidad. Finalmente en producción, estos devolverían las predicciones a las aplicaciones de los usuarios y almacenarían el live data en la instancia de la base de datos dedicada a albergar dicha información. Toda esta información es accesible al usuario Data Scientist a través de un Dashboard de monitorización.

El diseño del dashboard de monitorización propuesta sería la mostrada en la Fig. 32. El contenido de cada uno de los campos de la aplicación se genera de forma dinámica en base a los cambios que se produzcan en Cassandra y las acciones del usuario en la aplicación; este tipo de componentes se denominan componentes reactivos. El framework Dash, utilizado para desarrollar la aplicación web, permite la creación de componentes reactivos mediante el uso de callbacks en las funciones que forman parte del controlador de la aplicación. Los componentes reactivos de la aplicación son los siguientes:

- Selector de modelo: Permite seleccionar el modelo en producción que se quiere visualizar en la aplicación web.
- Reentrenamiento: Reentrena el modelo seleccionado con todos los datos disponibles en Cassandra, es decir, se agregan los datos históricos junto con los datos live. Puesto que para cada modelo pueden existir varias versiones (A, B, C…) se tendrá en cuenta la versión seleccionada por el usuario.
- Selección de versión: Permite visualizar la evolución gráfica y las métricas numéricas de las diferentes versiones entrenadas de un mismo modelo.

- To Champion: Botón que permite convertir una versión challenger a champion, lo que se traduce en que las respuestas a las peticiones de los usuarios serán realizadas por la versión del nuevo modelo seleccionado.
- Lista actual de modelos champion y challenger.
- Selector de variable: Seleccionado un modelo y su versión, permite elegir entre las diferentes variables del modelo y mostrar su evolución. Existen dos tipos de variables a mostrar:
    - Inputs: Variables de entrada del modelo. Se muestra una evolución de los valores que ha tomado la variable tanto en el conjunto histórico como el conjunto live.
    - Output: Variable que se quiere predecir. Para este tipo de variables se muestra la comparativa entre los valores reales y predichos en los conjuntos histórico y live.
- Gráfico de monitorización: Seleccionado un modelo, versión y variable que se quiere visualizar se muestra el gráfico de la evaluación o comparación de valores predichos y reales; dependiendo del tipo de la variable seleccionada.
- Métricas de monitorización para el historical data y live data: Se muestran dos tablas con los valores numéricos de las métricas de monitorización, que dependerán en si el modelo seleccionado es de tipo clasificación o regresión.

Por último, que el dashboard de monitorización se actualiza 2 veces por minuto para realizar un seguimiento de los cambios en la base de datos Cassandra.

En el apartado de implementación se verá la estructura de la aplicación de monitorización con el contenido de dos modelos de Machine Learning desplegados en producción.

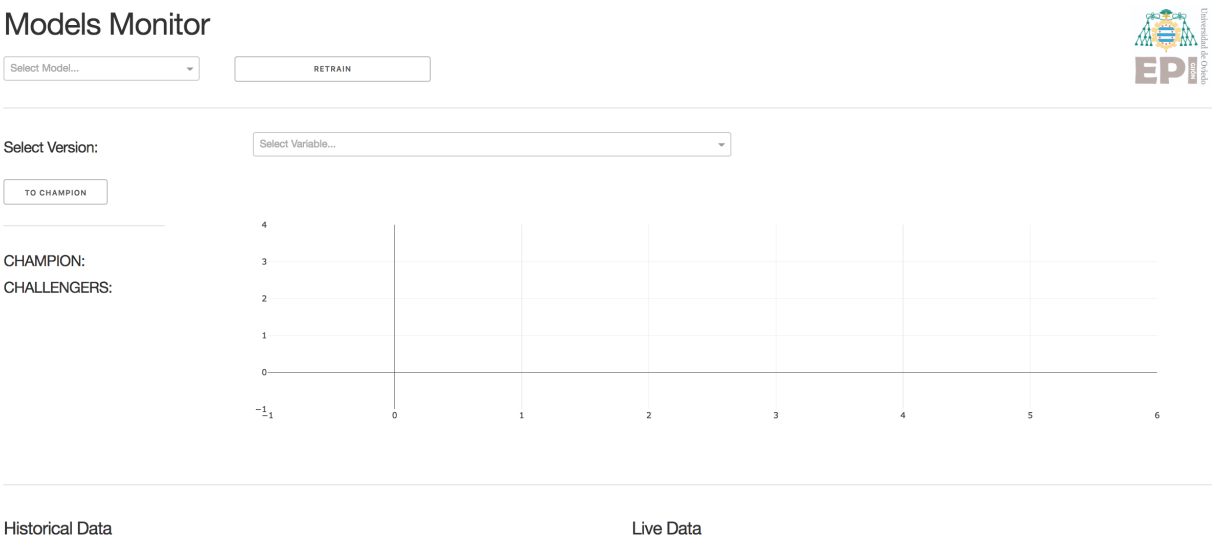


Figura 32. Esquema dashboard

Definidos los componentes del sistema del diagrama de diseño, en el siguiente apartado se explicará de manera detallada qué etapas realizar definiendo una metodología propia para el caso de estudio del proyecto.

6.- Propuesta de metodología

En este apartado se define la metodología a seguir para la correcta ejecución de los proyectos de modelado de datos realizados sobre la arquitectura expuesta anteriormente.

En la actualidad existen numerosas metodologías para llevar a cabo proyectos de modelado o minería de datos. Normalmente estas metodologías indican qué hacer en cada etapa del ciclo de vida, pero no cómo. Como resultado, cada organización acaba por adoptar una metodología propia, o incluso trabajar sin tener definidos los paso a seguir en cada etapa del proyecto. Esto se traduce en un aumento del tiempo necesario para llevar a cabo la ejecución del proyecto y como consecuencia pérdidas económicas, ya que ante nuevos proyectos se deben replantear de nuevo las etapas a realizar.

La metodología que se expone en este documento surge como una extensión de una de las metodologías más conocidas en la gestión de proyectos de minería de datos, CRISP-DM [34].

Utilizando como punto de partida el modelo de proceso que CRISP-DM define, se actualizará cada una de las etapas de los procesos que conforma para permitir su aplicación sobre la arquitectura basada en microservicios definida anteriormente.

En la Fig. 29 se ha mostrado el diagrama de diseño de los componentes que forman parte del sistema, junto con las etapas que engloban a cada uno de los componentes (parte superior de la figura). Indirectamente estas etapas están relacionadas con las fases generales que CRISP-DM define (Fig. 33).

Como se puede observar CRISP-DM adopta un flujo de trabajo flexible que permite volver a fases anteriores del ciclo si fuera necesario. Dentro de cada fase general existen tareas específicas, donde se describen las acciones a realizar para cada situación.

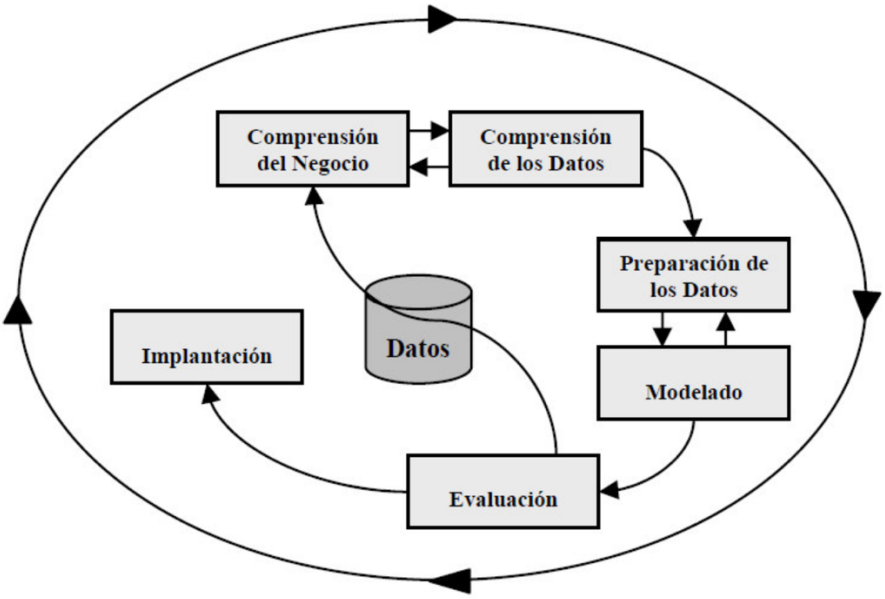


Figura 33. Modelo de proceso CRISP-DM

Como se ha señalado anteriormente, partiendo de esta información, extenderemos cada una de las fases dando lugar nuevas tareas que se adapten a las nuevas necesidades del proyecto, como por ejemplo el desarrollo de múltiples versiones de modelos. Por lo tanto, para cada fase del ciclo de vida de CRISP-DM el resultado sería:

- Comprensión del negocio: Esta fase involucra una serie de tareas de carácter teórico relacionada con la obtención de los objetivos del negocio, comprensión, evaluación de la situación y determinar los objetivos del proyecto de modelado. Como el sistema planteado no interfiere con esta fase previa. No se añadirán nuevas tareas.
- Comprensión de los datos: Comienza con la recolección de los datos del proyecto y almacenamiento en un sistema que proporcione acceso a los mismos para su posterior análisis. Para cubrir este punto bastaría con definir un sistema con una base de datos que actúe a modo de datalake para los

proyectos de modelado y proporcione acceso para realizar las consultas que se requieran.

A partir de este punto comenzaría el análisis de los datos. Las siguientes tareas a realizar según CRISP-DM consisten en *describir, explorar y verificar la calidad de los datos*. Para poder llevarlas a cabo, en una arquitectura orientada a microservicios, previamente sería necesario definir y crear los contenedores que albergarán cada servicio, por ello añadiremos dos nuevas tareas en esta fase:

Elaboración de imágenes base: En el servidor dedicado a proporcionar capacidad de cómputo deben estar definidas la o las imágenes base que darán lugar a los contenedores sobre los que se llevará a cabo el análisis de datos y entrenamiento de los modelos del proyecto. Si ya existen, debido a que se han creado para otros proyectos, se reaprovechan, si por el contrario no hay una imagen que proporcione un entorno para ejecutar código de manera remota en un lenguaje de programación determinado, se debe de crear.

Por imagen base se entiende a una plantilla que proporciona el esquema de creación de contenedores que permiten dotar al usuario de capacidad de cómputo remoto en el servidor, en un lenguaje de programación concreto y que contiene las librerías necesarias para comenzar con la fase del análisis minimizando el tiempo de puesta en marcha del entorno.

Para facilitar el acceso a la ejecución de imágenes, estas se deben de almacenar en un registro dedicado para su posterior distribución.

Despliegue de contenedores: A partir de las de las imágenes base almacenadas en el registro del servidor, se desplegarán aquellas que serán necesarias en el desarrollo del proyecto, dando lugar a uno o varios contenedores donde cada usuario puede acceder y comenzar con primeras tares de compresión de datos que CRISP-DM define. Estos contenedores se denominarán **microservicios de desarrollo**.

- Preparación de los datos: En esta fase CRISP-DM tareas de *selección, limpieza, estructuración (feature engineering), integración y formateo de los datos*. Estas tareas son perfectamente reutilizables dentro de la nueva propuesta y se llevarían a cabo accediendo a los microservicios de desarrollo que se desplegaron en la tarea de *Despliegue de contenedores* de la fase anterior. Se recomienda que todo el código generado en esta fase se ubique bajo un mismo directorio dedicado exclusivamente a las tareas de análisis que se llevan a cabo en las fases de *comprensión y preparación de los datos*.

  Por último, en la fase de *preparación de los datos* es necesario definir una última tarea extra que indique que se deben de almacenar el conjunto de datos resultante de realizar el análisis, para que se accesible en la siguiente fase de modelado.

  Almacenamiento del conjunto de entrenamiento: Una finalizadas las tareas anteriores, se tiene como resultado un conjunto de datos preparado para ser modelado. Con el objetivo de hacerlo accesible a los diferentes miembros del equipo y de cara a dar la posibilidad de que otras personas realicen el modelado en lenguajes diferentes al del análisis, se debe almacenar este en una en una instancia de una base de datos que denominaremos **Base de datos histórica**. La BD histórica almacena para cada proyecto los conjuntos de entrenamiento que posteriormente son utilizados en las fases de modelado. Esto permite numerosas ventajas:
  - Desacopla el flujo de trabajo entre análisis de datos y modelado, lo que permite a gente especializada en diferentes fases dividir el trabajo de manera metodológica.
  - Utilizar diferentes lenguajes de programación para el análisis y modelado, gracias a que el conjunto de datos de entrenamiento se almacena en una base de datos accesible desde cualquier lenguaje.

- Siempre que se sigan las etapas de la metodología, cada miembro del equipo puede usar el lenguaje en el que sienta más cómodo y tenga una mayor experiencia. No se obliga a utilizar un lenguaje específico por proyecto.

- Modelado: La fase de modelado consiste en seleccionar la técnica de modelado más adecuada para un proyecto de minería de datos específico. Se recomienda, al igual que en la fase de análisis, crear un directorio de trabajo para las diferentes tareas que se van a realizar en esta fase.
  En la fase anterior se introdujo la tarea de *almacenamiento del conjunto de entrenamiento*, la primera tarea que necesitamos definir, y que CRISP-DM no contempla, es la obtención del conjunto de datos de entrenamiento.

  Obtención del conjunto del conjunto de entrenamiento: Consiste en acceder a la instancia de la BD histórica y obtener el conjunto de entrenamiento.

  En este punto ya se puede comenzar con las tareas que CRISP-DM proporciona para guiar la fase de modelado: *Selección de la técnica de modelado, generación del plan de prueba, construcción del modelo, y evaluación*. Todas ellas se seguirán realizando a través del microservicio de desarrollo.
  Una de las carencias más importantes de CRISP-DM es que solo contempla la posibilidad de implementar un solo modelo resultante de la fase de modelado. Por lo tanto es necesario extender y definir la metodología para que abarque la inclusión y puesta en producción de más de un modelo de datos. Para ello definimos la siguiente tarea:

  Modelado de múltiples prototipos: Si se desea desarrollar y poner en producción más de un modelo, se deberá realizar las tareas anteriores que

CRISP-DM define para cada versión que queramos desplegar. Como resultado tendremos *n* modelos, los cuales etiquetaremos con los caracteres A, B, C… o con identificadores numéricos, según se prefiera. Posteriormente se crearan tantos directorios como versiones, dichos directorios contendrán los archivos necesarios para la puesta en producción de cada uno de ellos; estos directorios serán denominados **directorios de despliegue**.
Para cada versión de modelo generada se deberá realizarán las siguientes nuevas tareas:

Almacenamiento de predicciones: Calcular los valores predichos de cada modelo sobre el conjunto de entrenamiento. Estas predicciones se almacenarán en la BD histórica junto a los datos de entrenamiento y permitirá ver la calidad obtenida en el conjunto de datos de entrenamiento. Como resultado habrá una columna de predicciones por cada versión del modelo junto con los valores reales. Esto permitirá comparar los resultados desde el sistema de monitorización.

Serialización de los modelos: Se debe salvar la versión entrenada de cada uno de los modelos A, B, C… La forma de serializar cada modelo depende del lenguaje de programación en el que se hallan desarrollado. Cada versión del modelo será almacenado en su directorio de despliegue correspondiente, creado en la tarea de *modelado de múltiples prototipos*.

Por último, se debe definir como se van a almacenar las variables que el usuario del modelo envíe para obtener predicciones. Este tipo de datos, que son enviados cuando el modelo se encuentren en producción se denominarán *live data.* Para almacenar este tipo de información se creará una nueva instancia en la base de datos, aparte de la BD histórica, denominada **base de datos live**. La tarea encargada de realizar esta labor es la siguiente:

Definición de live data: Consiste en crear en la BD live las tablas necesarias para almacenar los datos enviados por el usuario, la predicción realizada por el modelo y el valor real, con el objetivo de emplear esta información posteriormente en las métricas de rendimiento.

A diferencia de la BD histórica donde para cada conjunto de datos de entrenamiento se creaban nuevas columnas por cada predicción de las diferentes versiones del modelo, en la BD live existirá una tabla por cada versión del modelo. Este cambio en el modelo de datos de la BD, es debido a que diferentes modelos pueden requerir diferentes variables de entrada, al crear una tabla por cada versión tendremos un registro de que variables son empleadas por cada versión.

- Evaluación: Esta fase consiste en realizar una *evaluación de los resultados* y *proceso de revisión,* como indica CRISP-DM, en relación con los procesos de negocio. Si los resultados son satisfactorios debemos emplear una tarea adicional en la que registremos las versiones de los modelos desarrollados finalmente.

  Registro de modelos: Consiste en proporcionar una descripción de los diferentes modelos en producción. Para ello se recomienda crear una tabla en la BD histórica en la cual se describa:
  - Nombre del proyecto al que pertenece el modelo.
  - Versión: A, B, C…
  - Modo: Champion o challenger. Si el modelo es champion será el que devuelva la predicción al usuario y almacenará el live data recibido en la BD live, si es challenger únicamente almacenará el live data en BD live.
  - URL: Dirección desde la cual será accesible el modelo, una vez se encuentre en producción.

Si la tabla ya existe debido a que se ha creado para otro proyecto, se reaprovecharía su estructura.

- Implementación: Es la última fase de la metodología. Consiste en describir como se han de poner en producción los modelos generados. CRISP-DM indica en sus dos tareas *plan de implementación e informe final* que el modelo debe ser monitorizado y mantenido a lo largo del tiempo y que finalmente se debe realizar un informe final como conclusión del proyecto. Como ampliación de esta última fase, proponemos las siguientes tareas adicionales:

  Creación de API´s Rest: Para hacer accesible cada uno de los modelos a los usuarios o aplicaciones de usuario es necesario definir una API por cada versión del modelo con el objetivo de que el usuario final pueda consumir los servicios de predicción. Estas deberán encontrarse en el mismo directorio que el modelo serializado resultante de la tarea de *serialización de los modelos*. Se recomienda crear una función adicional a la de predicción en la API que permita reentrenar el modelo con los datos disponibles tanto en la BD histórica y la BD live, con el objetivo que desde una aplicación de monitorización se pueda automatizar este proceso.

  Despliegue de contenedores de producción: Para hacer accesible las API´s de los modelos se debe de crear un contenedor que contenga y exponga el contenido del directorio donde se ha almacenado el código de la API y el modelo serializado. Para ello se volverá a utilizar la tecnología de contenedores en donde a partir de una imagen definida en el directorio de despliegue se creará un contenedor que expone la función de predicción de la API. Tendremos finalmente tantos contenedores como versiones de modelos queramos poner en producción. Este segundo tipo de contenedores serán

denominados **microservicios de producción** y estarán accesibles en el registro de imágenes del sistema para su ejecución**.**

Monitorización: La tarea de monitorización debe permitir controlar la calidad de los modelos que se encuentran en producción. CRISP-DM no profundiza en este aspecto, únicamente señala que se debe realizar una monitorización y mantenimiento para que la calidad de las predicciones que estos ofrecen al usuario sea adecuada.

En un sistema de producción basado en microservicios, en donde cada versión del modelo es un servicio independiente que escribe en una base de datos los datos recibidos y devuelve una predicción, tenemos información suficiente para crear una aplicación que permita un control de la calidad de los mismos. Lo más adecuado en estos casos es desarrollar un dashboard interactivo que permita observar diferentes parámetros de las diferentes versiones de los modelos.

Se propone, por tanto, el desarrollo de una aplicación web con un diseño tipo dashboard que permita, como mínimo, la visualización y monitorización de:

- Los diferentes modelos y versiones que se encuentran en producción.
- Información de, paraca cada proyecto, que modelo se encuentra en modo champion y challenger en cada momento. Además de permitir convertir versiones challenger a champion.
- Evolución de los valores de las variables en el tiempo.
- Comparativa entre valores predichos y reales de las variables que se desean predecir.
- Métricas de evaluación de modelos. Como para cada tipo de modelo (clasificación, regresión…) las métricas cambian significativamente se debe realizar un estudio de cuales son las mejores para el seguimiento en cada caso.

- Capacidad de reentrenamiento de un determinado modelo, si se ha implementado una función específica para funcionalidad en la tarea de *creación de API´s Rest*.

Finalmente, el diagrama de la metodología CRISP-DM resultante es el que se muestra en la Fig. 34. Las tareas señaladas con los caracteres (*) indican que son las nuevas tareas creadas sobre las tareas de CRISP-DM.

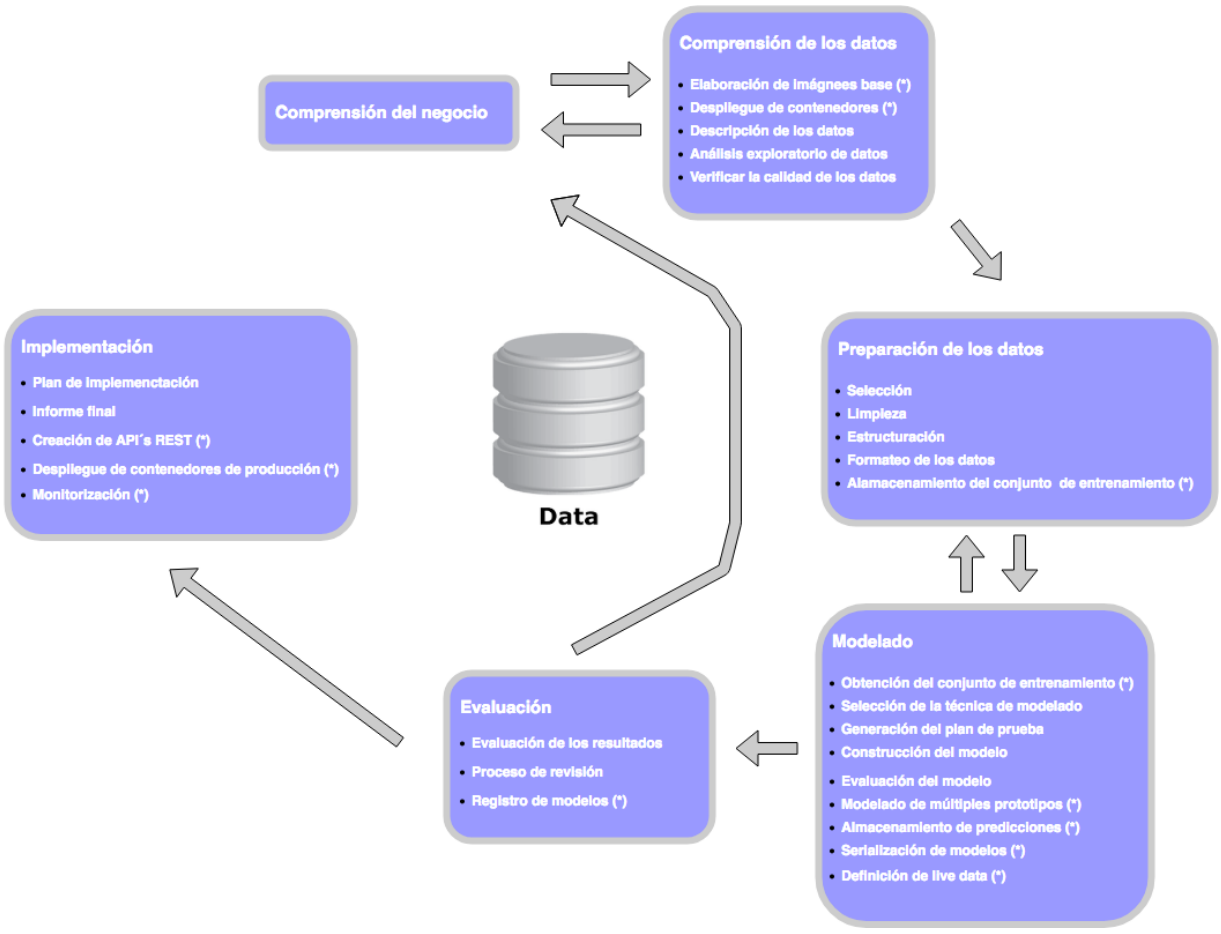


Figura 34. Metodología de integración

7.- Implementación

Con el objetivo de ilustrar la metodología propuesta, en este apartado se presentarán dos casos de uso acerca de dos problemas de modelización diferentes, a resolver desde el punto de vista del rol de Data Science. Ambos desarrollos se realizarán sobre un sistema que proporciona microservicios de desarrollo y de producción, el cual sigue la misma estructura que el de la arquitectura propuesta.

El código necesario para crear las imágenes Docker empleadas en la implementación viene dado en el anexo número uno del presente documento. Junto a la memoria del proyecto también se proporcionan dos directorios con los datasets y código R y Python resultante de aplicar la metodología, así como el código del dashboard de monitorización empleado. Con el objetivo de no extender demasiado esta ilustración, en las explicaciones sucesivas no se entrará en detalles del código de los scripts desarrollados, únicamente se nombrarán las funciones que realiza cada uno.

Los dos problemas a modelar estarán desarrollados en R y Python para ejemplificar las posibilidades que ofrece el sistema.

- El desarrollo en R tendrá como finalidad modelizar un problema de aprendizaje supervisado de clasificación multiclase. Se utilizará como dataset de prueba “Iris” y se realizarán dos tipos de modelos A y B que se pondrán en producción. El modelo A realiza la clasificación en base al algoritmo SVM (Support Vector Machine) con un kernel de tipo radial, el modelo B estará entrenado tipo SVM pero con un kernel tipo polinómico.

  El dataset Iris contiene 3 clases con 50 instancias cada una, donde cada clase pertenece a un tipo de planta iris. El objetivo es predecir a que tipo de clase pertenece una planta iris dadas la longitud y anchura de sus pétalos y sépalos.
- El desarrollo en Python modelizará un problema de aprendizaje supervisado de regresión. Se utilizará como dataset base de prueba “Wine Quality”. Al igual que el modelo de clasificación, se realizarán dos tipos diferentes de modelos cuyas versiones se denominarán A y B. La versión A implementa un algoritmo

Random Forest, la versión B está basada en un algoritmo SVR (Support Vector Regression) el cual es una extensión del modelo SVM para aplicar regresión.

El dataset Wine Quality contiene información acerca de características fisicoquímicas junto con la calidad de resultante del vino en una escala de números enteros con valoración de 3 a 8. El objetivo es predecir el valor de la calidad en base a sus características; se pueden emplear tanto técnicas de regresión como de clasificación, en este caso se ha optado por la técnica de regresión.

7.1.- Sistema de producción

El sistema de producción se ha desarrollado en una máquina virtual con las siguientes características técnicas del sistema de producción son:

- S.O: Ubuntu Xenial 16.04.4 LTS.
- CPU: 2,2 GHz Intel Core i7.
- RAM: 2GB.
- Almacenamiento: 10GB HDD.

El software base que precisa el sistema es Docker Engine Community v.18.03.0.

El siguiente paso sería desplegar los microservicios necesarios para realizar el desarrollo remoto, almacenamiento de imágenes, almacenamiento de datos, despliegue en producción y monitorización. Para ello se emplean las siguientes imágenes Docker:

- Mysql:5.5- Emulará un datalake el cual será accesible para obtener los datos de una BD MySQL.
- Template_p:3.6.0- Permitirá la ejecución remota de un intérprete Python desde Pycharm.

- Template_r:3.4.0- Despliega RStudio Server para el desarrollo remoto en el lenguaje R.
- Cassandra:2.1- Es el contenedor que se encargará del almacenamiento de las instancias histórica y live, junto con las tablas necesarias.
- Registry:2- Un contenedor dedicado como registro privado de imágenes Docker.
- Flask:monitor: Despliega el dashboard de monitorización.

A partir de estas imágenes podremos desplegar los contenedores base para proporcionar los microservicios necesarios para el desarrollo de cualquier modelo de datos. Los comandos bash necesario para llevar a cabo la puesta en marcha de las bases de datos de los contenedores y crear el modelo de datos de Cassandra son los siguientes:

```
# MySQL
$ sudo docker run --name datalake -d -p 3306:3306 mysql:5.5

# Cassandra
$ sudo docker run --name db -d -p 9160:9160 -p 9042:9042 cassandra:2.1

# Accedemos al interprete cqlsh para crear la estructura de nuestro
# sistema de almacenamiento
$ sudo docker exec -ti db bash

cqlsh> CREATE KEYSPACE historical_db
  ... WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

cqlsh> CREATE KEYSPACE live_db
  ... WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

Por último desplegamos el dashboard que servirá para monitorizar las futuras implementaciones:

```
$ sudo docker run --name monitor -d -p 18888:8888 flask:monitor
```

Con el objetivo de emular una situación real, el contenedor que actúa de datalake debería de contener los datos para comenzar con el desarrollo de ambos proyectos. Por este motivo, se han creado los scripts *upload_mysql.py* y *upload_mysql.R*. Su finalidad es leer los datos de los *csv´s* que contienen la información de ambos datasets y almacenar parte de su en el contenedor datalake. La otra parte será utilizado para emular live data (datos que envía el usuario para obtener una predicción). Aproximadamente un 20% del total de instancias del conjunto de entrenamiento será utilizado para emular live data.

En este punto ya podemos con desarrollo el de los modelo R y Python siguiendo la metodología propuesta.

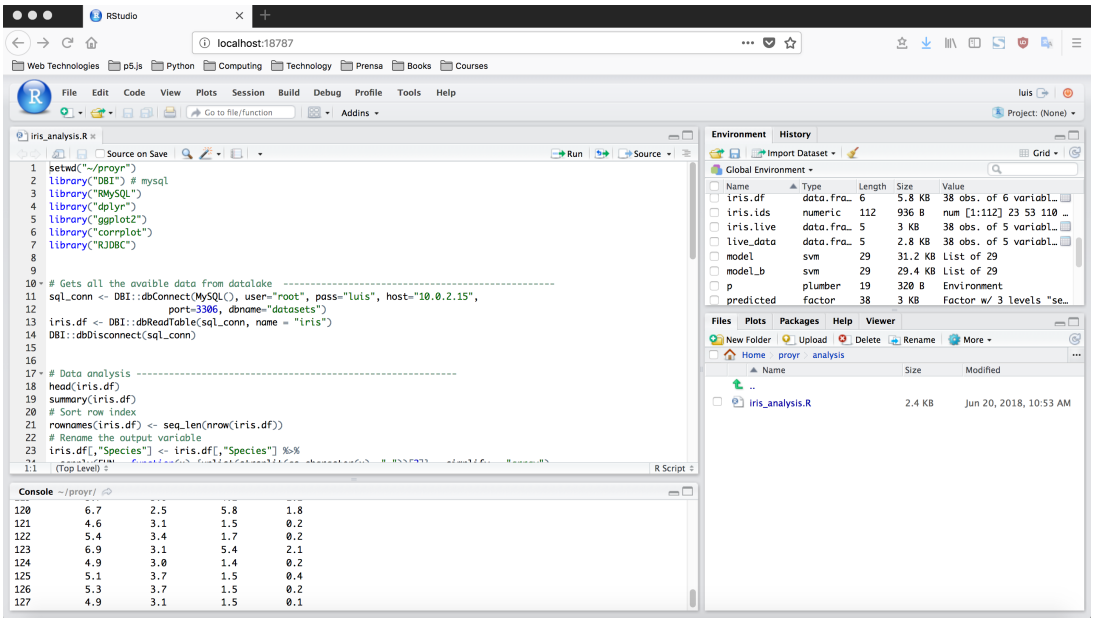
7.2.- Modelo R

Las fases involucradas han sido:

- Comprensión de los datos:
    - *Elaboración de imágenes base*: En la puesta en marcha del sistema de producción se han elaborado 2 imágenes base para dar soporte a proyectos R y Python. Por lo tanto esta tarea ya se ha llevado a cabo.
    - *Despliegue de contenedores*: En el servidor de producción se ejecuta la imagen template_r:3.4.0 montando un directorio local para que todo el código creado en el contenedor sea persistente.

```
# RStudio Server
$ sudo docker run --name proyr -d -p 8787:8787 \
    -v /home/luis/proyects/proyr:/home/rstudio/proyr \
    template_r:3.4.0
```

Accediendo a través de un navegador al puerto papeado en el servidor ya tendrías el entorno de desarrollo remoto disponible como se puede ver en la siguiente figura.



A continuación, dentro del directorio *proyr/analysis* se encuentra el script *iris_analysis.R* el cual contiene el código de llevar a cabo las tareas de *descripción de los datos, análisis exploratorio de los datos y verificar la calidad de los datos* correspondientes a esta fase.

- Preparación de los datos:

  Toda las tareas de la fase: *Selección, limpieza, estructuración, formateo de los datos y almacenamiento del conjunto de entrenamiento* se llevan a cabo dentro del script *iris_analysis.R.* El resultado de esta tarea crea una tabla denominada iris en Cassandra dentro de la instancia *historical_db*.

- Modelado:

  Todas las tareas relacionadas con la fase de modelado se llevan a cabo dentro del script iris_training.R en el directorio *proyr/training.* Como resultado se obtiene:

- Las dos versiones A y B serializadas del modelo de clasificación, ubicadas en sus respectivos directorios de despliegue denominados *proyr/deployA* y *proyr/deployB.*
- Modificación del contenido de la tabla iris dentro de la instancia historical_db en donde se han añadido dos nuevas columnas predicted_a y predicted_b correspondientes a la predicciones de ambos versiones sobre el conjunto de entrenamiento. Para ilustrar el contenido de la tabla iris resultante se muestra la siguiente imagen:

id	label	petal_length	petal_width	predicted_a	predicted_b	sepal_length	sepal_width
23	versicolor	3.7	1	versicolor	versicolor	5.5	2.4
53	virginica	5.6	2.4	virginica	virginica	6.3	3.4
110	virginica	5.4	2.3	virginica	virginica	6.2	3.4

- La definición de 2 nuevas tablas en la instancia *live_db* de Cassandra que contienen la estructura de los datos a recibir cuando los modelos se encuentren en producción. Ambas tablas se denominan iris_a e iris_b.
  Para emular que se ha habido actividad por parte del usuario, se ejecuta el script *upload_mysql.R*. De esta manera las tablas iris_a iris_b contendrán valores predichos y reales para ser monitorizadas posteriormente.

- Evaluación: Realizadas las tareas teóricas de *evaluación de los resultados y proceso de revisión*, se debe de crear el registro de modelos en la instancia historical_db.

```
cqlsh> CREATE COLUMNFAMILY model_registry
  ... (
  ... name text
  ... , url text
  ... , type text
  ... , mode text
  ... , version text
  ... , PRIMARY KEY (name, version)
  ... );
```

- Implementación:

  Para finalizar el ejemplo de implementación del modelo R, nos centraremos en las nuevas tareas que hemos definido sobre CRISP-DM

  - *Creación de API´s REST*: Dentro de los directorios deployA y deployB se encuentra el código de las API´s Rest en los archivos api.R y la definición de la IP y puerto de escucha de las mismas en el archivo plumber.R.

    Ambas API´s exponen las funciones de predicción y reentrenamiento. Una vez se ejecutan lo primero que realizan es registrarse en la tabla model_registry de Cassandra para hacerlas accesibles ante el dashboard de monitorización y aplicaciones de usuario.

```
 name | version | mode       | type           | url
------+---------+------------+----------------+----------------
 iris |       A |   champion | classification | localhost:18000
 iris |       B | challenger | classification | localhost:18001
```

  - *Despliegue de contenedores de producción*: Para el despliegue de las API´s Rest referentes a cada modelo, en el servidor por cada directorio deployA y deployB creados en esta implementación, se ejecutarían los siguientes comandos para construir una imagen:

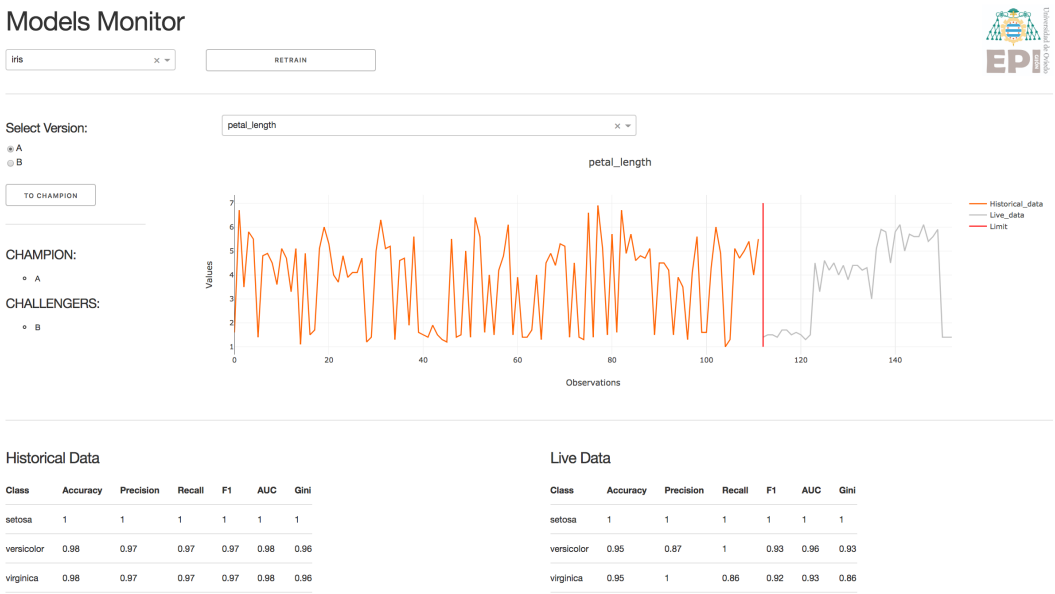
```
$ sudo docker build -t plumber:iris_a .
$ sudo docker build -t plumber:iris_b .
```

La cual ejecutaríamos para dar lugar a los contenedores, a través de los siguientes comandos.

```
$ sudo docker run --name iris_a -d -p 18000:18000 plumber:iris_a
$ sudo docker run --name iris_b -d -p 18001:18001 plumber:iris_b
```

En este punto el usuario ya puede realizar peticiones a los modelos de Machine Learning desplegados.

- *Monitorización*: A través del contenedor desplegado en la puesta en marcha del sistema de monitorización, podemos acceder al dashboard web el cual muestra, para el modelo iris, la siguiente vista:

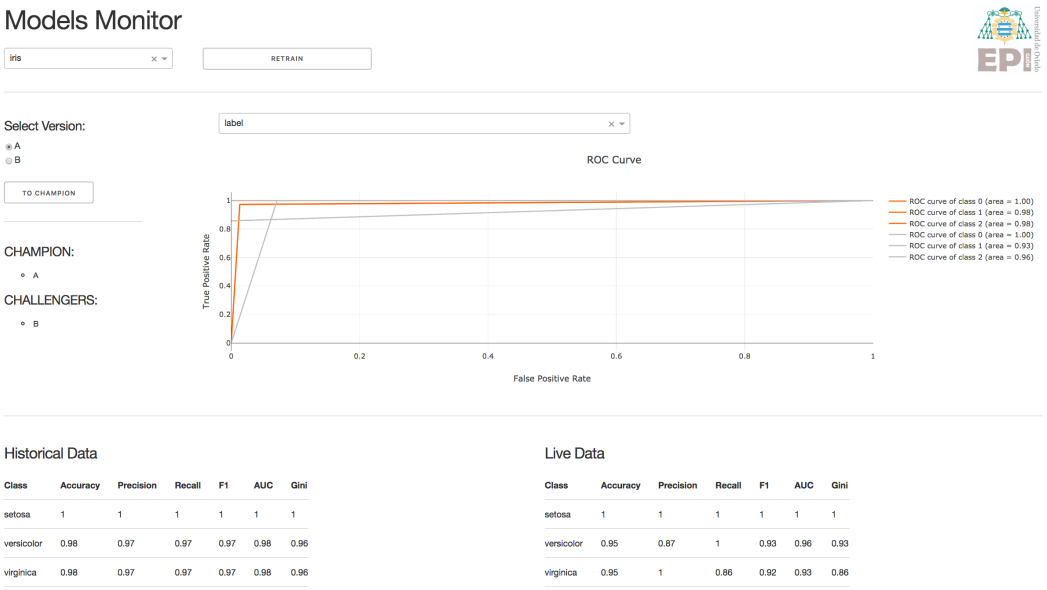


En la parte superior se muestra el **modelo seleccionado** iris junto con la opción de **reentrenamiento**.

Seguidamente podemos elegir entre las **versiones del modelo** que han sido desarrolladas; en este caso han sido dos versiones, A y B y la opción de convertir una u otra **a modo *champion***. En la parte inferior se

muestran dos listas que indican, actualmente quién es el ***champion*** y quién o quienes son los ***challengers***. En la parte derecha se puede visualizar, por un lado el **selector de variables** a mostrar en el **gráfico** inferior. Para cada input se muestra la evolución de la variable tanto en el conjunto de datos histórico, como en el conjunto de datos live. Para la variable output, en el caso del actual modelo de clasificación se muestra el gráfico de la curva ROC.

La curva ROC presenta el ratio de verdaderos positivos en el eje Y y el ratio de falsos positivos en el eje X. Esto significa que la esquina superior izquierda del gráfico ROC representa el punto ideal (un ratio de falsos positivos de cero y un ratio de verdaderos positivos de uno). El objetivo ideal es maximizar el área bajo la curva (AUC) maximizando el ratio de positivos reales y minimizando la tasa de positivos falsa.



Class	Accuracy	Precision	Recall	F1	AUC	Gini
setosa	1	1	1	1	1	1
versicolor	0.98	0.97	0.97	0.97	0.98	0.96
virginica	0.98	0.97	0.97	0.97	0.98	0.96

Class	Accuracy	Precision	Recall	F1	AUC	Gini
setosa	1	1	1	1	1	1
versicolor	0.95	0.87	1	0.93	0.96	0.93
virginica	0.95	1	0.86	0.92	0.93	0.86

En la parte inferior de la aplicación web se encuentran dos tablas con valores numéricos referentes a las **métricas de monitorización** definidas para los problemas de clasificación. Una muestra las métricas

para los valores definidos en los datos históricos (training) y la otra muestra las métricas de los datos live (test), para cada clase de etiqueta.

La estructura de directorios y el código generado resultante es:

```
|- proyr
  |- data
    |- iris.csv
  |- analysis
    |- iris_analysis.R
  |- training
    |- iris_training.R
  |- deployA
    |- iris_modelA.Rds
    |- plumber.R
    |- api.R
    |- Dockerfile
  |- deployB
    |- iris_modelB.Rds
    |- plumber.R
    |- api.R
    |- Dockerfile
  upload_livedata.R
  upload_mysql.R
```

7.3.- Modelo Python

La metodología que se aplica a la hora de desarrollar el segundo caso de modelización es la misma que el caso anterior a excepción del aspecto tecnológico que implica el uso de un lenguaje diferente al anterior. Las fases involucradas son:

- Comprensión de los datos:
    - *Elaboración de imágenes base*: Ya existe una imagen, elaborada en la instalación del sistema de producción, denominada template_p que cubre esta tarea.
    - *Despliegue de contenedores*: A diferencia de el caso de uso desarrollado en R, en Python el IDE se ejecute localmente en el equipo del usuario. A través de Pycharm habilitamos la ejecución de intérprete remoto conectándonos a la imagen template_p y creando un contenedor por cada ejecución, el cual es eliminado tras retornar el resultado al usuario.

La finalidad del intérprete remoto es permitir la ejecución remota de tareas que requieren una capacidad de cómputo mayor que la del equipo del usuario, por ejemplo la fases de entrenamiento de los modelos.

Las tareas de *descripción de los datos, análisis exploratorio de los datos y verificación la calidad de los datos* se encuentran en el script wine_analysis.py dentro del directorio *proyp/analysis*.

- Preparación de los datos:

  El archivo wine_analysis.py contiene el resultado de aplicar las tareas de *selección, limpieza, estructuración, formateo de los datos y almacenamiento del conjunto de entrenamiento.* El resultado de esta tarea da lugar una tabla denominada *wine* en Cassandra dentro de la instancia *historical_db*.
- Modelado:

  Todas las tareas relacionadas con la fase de modelado se llevan a cabo dentro del script *wine_training.py* en el directorio *proyp/training.* En esta tarea es donde entra en juego la ejecución remota y su importancia. La fase de análisis se puede ejecutar, como norma general, en equipos con bajas prestaciones, sin embargo, en las fases de entrenamiento donde se realizan ajustes de hiperparámetros de los modelos a través de métodos, como por ejemplo, Grid Search, es necesario hacer las ejecuciones en servidores de altas prestaciones para reducir el tiempo de espera hasta tener un modelo entrenado de cada versión.

  Al igual que en ejemplo anterior, se obtiene como resultado:

  - Las dos versiones A y B serializadas del modelo de regresión, ubicadas en sus respectivos directorios de despliegue denominados *proyp*/*deployA* y *proyp/deployB.*
  - Modificación del contenido de la tabla *wine* dentro de la instancia historical_db en donde se han añadido dos nuevas columnas

predicted_a y predicted_b correspondientes a las predicciones de ambos versiones sobre el conjunto de entrenamiento.

- La definición de 2 nuevas tablas en la instancia *live_db* de Cassandra que contienen la estructura de los datos a recibir cuando los modelos se encuentren en producción. Ambas tablas se denominan wine_a y wine_b.

- Evaluación: Realizadas las tareas teóricas de *evaluación de los resultados y proceso de revisión*, se debe de crear el registro de modelos en la instancia historical_db.
- Implementación:

  Para finalizar el ejemplo de implementación del modelo python, nos centraremos en las nuevas tareas que hemos definido sobre CRISP-DM

  - *Creación de API´s REST*: Dentro de los directorios deployA y deployB se encuentra el código de las API´s Rest y definición de IP y puerto de escucha en los scripts *app.py*.

    Ambas API´s exponen las funciones de predicción y reentrenamiento. Una vez se ejecutan lo primero que realizan es registrarse en la tabla model_registry de Cassandra para hacerlas accesibles.
  - *Despliegue de contenedores de producción*:

    Construimos los microservicios de producción a partir de los Dockerfile definidos en los directorios deployA y deployB.

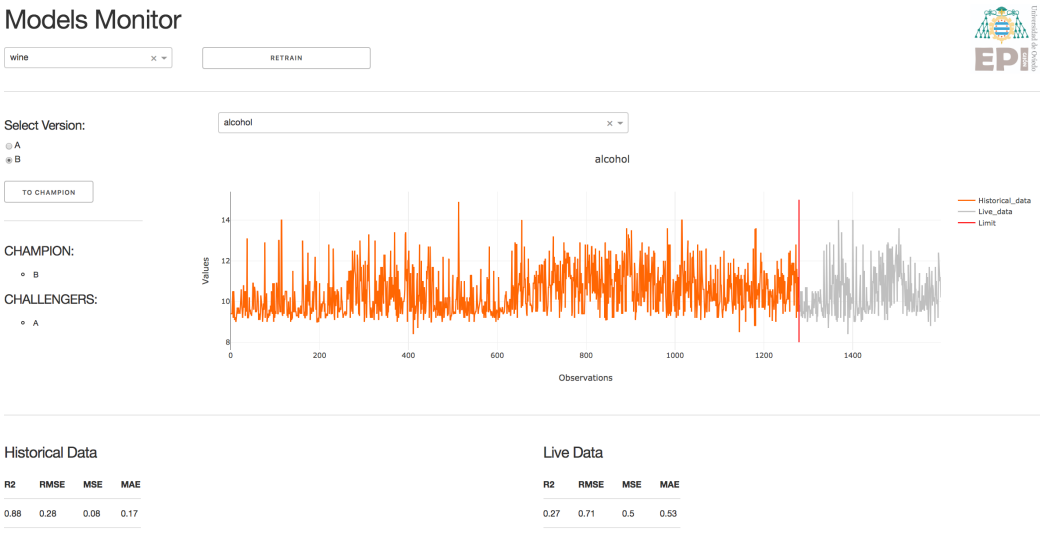
```
$ sudo docker build -t flask:wine_a .
$ sudo docker build -t flask:wine_b .
```

    Ejecutamos ambas imágenes

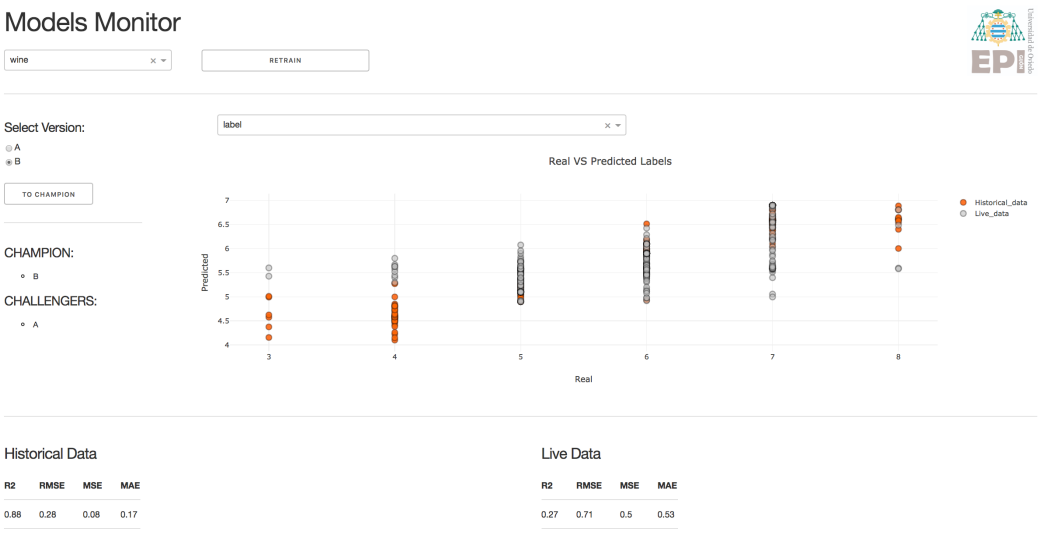
```
$ sudo docker run --name wine_a -d -p 19000:19000 flask:wine_a
$ sudo docker run --name wine_b -d -p 19001:19001 flask:wine_b
```

  - *Monitorización*:

La lógica de la aplicación es la misma que en el caso del anterior expuesto acerca del modelo R. En la parte superior de la aplicación se encuentra el modelo seleccionado, wine, junto con la opción de reentrenamiento. En la parte inferior se encuentra la versión del modelo que se está visualizando gráfica y numéricamente. Se puede observar como en este caso la versión B desarrollada está actuando en modo champion, lo que significa que el microservicio de producción de la versión B del modelo es la que está proporcionando las predicciones a las peticiones de usuario.



Al igual que en la implementación del modelo de clasificación, al seleccionar la variable output se muestra una comparativa entre los valores predichos y reales. El caso de la curva ROC era específico para los modelo de clasificación, en el caso de la regresión se presenta un gráfico tipo scatter plot.



La estructura de directorios y el código generado resultante es:

```
|- proyp
  |- data
    |- wine_quality.csv
  |- analysis
    |- wine_analysis.py
  |- training
    |- wine_training.py
  |- deployA
    |- wine_modelA.pkl
    |- app.py
    |- Dockerfile
  |- deployB
    |- wine_modelB.pkl
    |- app.py
    |- Dockerfile
  upload_livedata.py
  upload_mysql.py
```

8.- Presupuesto

En este apartado se lleva a cabo una medición de los gastos que conlleva la ejecución del proyecto.

8.1.- Capítulo 1: Recursos humanos

Identificador	Perfil	Salario (euros/hora)	Perfiles	Horas	Presupuesto (Euros)
1.1	Jefe de proyecto	18,00	1	776	13.968,00
1.2	Analista-programador	14,00	2	776	21.728,00
1.3	Administrador de sistemas	10,00	2	168	3.360,00
Total:					39.056,00

8.2.- Capítulo 2: Hardware

Como servidor de cómputo y de puesta en producción de los modelos Machine Learning resultantes de aplicar la metodología descrita en este proyecto, se propone un servidor con las siguientes características:

Identificador	Componente	Unidades	Precio/Ud (Euros)	Presupuesto (Euros)
2.1	HPE ProLiant DL80 Gen9 • Tamaño: 2U • Sockets: 2 • Nº GPUs: 1	1	2.057,00	2.057,00
2.2	Procesador Intel Xeon E5-2620 v4 • Núcleos 18	1	439,90	439,90
2.3	Memoria HP 8 GB • DD4 • 2400 MHz	3	280,90	842,70
2.4	Almacenamiento HPE 10TB • SATA 7.2K rpm • LFF	1	986,80	986,80
Total				4.326,40

8.3.- Capítulo 3: Licencias software

El número de licencias software a adquirir depende directamente del número de personas implicadas en el uso de la futura solución. Se estima que con 10 licencias de Pycharm se cubrirá la demanda para futuros desarrollos.

Identificador	**Descripción**	**Cantidad**	**Precio/Ud (Euros)**	**Presupuesto (Euros)**
3.1	Pycharm	10	195,00	1.950,00
Total:				1.950,00

8.4.- Resumen del presupuesto

En la siguiente tabla se muestra el presupuesto por ejecución.

Concepto	**Importe (Euros)**
Recursos humanos	39.056,00
Hardware	4.326,40
Licencias software	1.950,00
Total:	45.332,40

Al presupuesto de ejecución se le debe añadir los costes indirectos asociados al proyecto y el IVA.

Concepto	**Importe (Euros)**
Presupuesto de ejecución	45.332,40
Costes indirectos (10%)	4.533,24
Total sin IVA:	49.865,64
IVA (21%)	10.471,78
Total:	60.337,42

9.- Planificación

En este apartado se definen las tareas que conforman la planificación necesaria para desplegar los componentes requeridos para la aplicación de la metodología presentada en este proyecto. Dentro de la planificación se diferencian 3 grupos de tareas que tienen finalidades diferentes.

Por un lado se encuentra el grupo de despliegue técnico del sistema de producción. La finalidad de estas tareas es adquirir e instalar el hardware necesario para desplegar las bases datos y el software de gestión de contenedores, Docker.

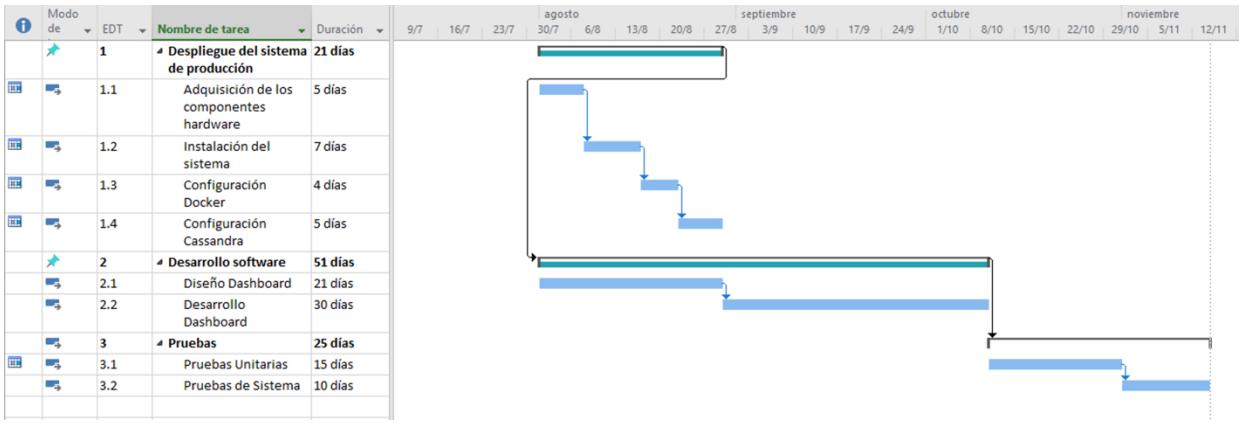
Por otro lado se encuentra la parte de desarrollo software. Este segundo grupo de tareas involucran las etapas de desarrollo del dashboard de monitorización, las cuales pueden comenzar a realizarse en paralelo con el primer grupo de tareas.

Por último se encuentra el grupo de tareas referentes a las pruebas. Una vez el sistema se encuentre operativo y el dashboard de monitorización haya sido desarrollado, se deben de realzar las pruebas software necesarias con el objetivo de verificar el correcto funcionamiento de la aplicación y su correcto

La composición de tareas de la planificación sería la siguiente:

EDT	Tarea	Inicio	Fin	Días
1	Despliegue del sistema de producción	01/08/2018	29/08/2018	21
1.1	Adquisición de los componentes hardware	01/08/2018	07/08/2018	5
1.2	Instalación del sistema	08/08/2018	16/08/2018	7
1.3	Configuración Docker	17/08/2018	22/8/2018	4
1.4	Configuración de Cassandra	23/08/2018	29/08/2018	5
2	Desarrollo software	01/08/2018	10/10/2018	51
2.1	Diseño Dashboard	01/08/2018	29/08/2018	21
2.2	Desarrollo Dashboard	30/08/2018	10/10/2018	30
3	Pruebas	11/10/2018	14/11/2018	25
3.1	Pruebas Unitarias	11/10/2018	31/10/2018	15
3.2	Pruebas de Sistema	01/11/2018	14/11/2018	10

El diagrama Gantt resultante de la planificación es:



Modo de	EDT	Nombre de tarea	Duración
	1	**Despliegue del sistema de producción**	**21 días**
	1.1	Adquisición de los componentes hardware	5 días
	1.2	Instalación del sistema	7 días
	1.3	Configuración Docker	4 días
	1.4	Configuración Cassandra	5 días
	2	**Desarrollo software**	**51 días**
	2.1	Diseño Dashboard	21 días
	2.2	Desarrollo Dashboard	30 días
	3	**Pruebas**	**25 días**
	3.1	Pruebas Unitarias	15 días
	3.2	Pruebas de Sistema	10 días

Figura 35. Diagrama Gantt

10.- Conclusiones

El objetivo inicial de la ejecución de este proyecto ha sido el estudio del estado del arte de plataformas que permitan el despliegue de modelos Machine Learning atendiendo a las necesidades que se presentaron en el alcance del proyecto. Como no existe ninguna tecnología que cubra dichos requisitos se optó por el diseño de un sistema que permita el desarrollo remoto y puesta en producción de los modelos basados en datos en base a microservicios que los miembros del equipo pueden crear y utilizar según sus necesidades. Para definir de manera metodológica como gestionar los proyectos de minería de datos en base a esta nueva aproximación, se ha propuesto una metodología que parte de CRISP-DM extendiendo el número de tareas de esta para indicar qué y cómo se ha de realizar cada paso del ciclo de vida del proyecto de modelado de datos.

Para ilustrar los conceptos teóricos expuestos, se ha realizado una implementación acerca de la resolución de dos problemas de Machine Learning académicos, con el objetivo de exponer cada una de las fases de la metodología y como se relacionan estas con los microservicios definidos.

Por último, se ha desarrollado un prototipo de dashboard que permite la visualización de diferentes métricas de monitorización, comportamiento de las variables, predicciones, cambio de entre modos champion y challenger de las diferentes versiones de los modelos en producción y reentrenamiento de cualquier versión con todos los datos disponibles hasta el momento.

Debido a la diversa pluralidad de modelos de aprendizaje existentes hoy en día. La solución planteada en este proyecto no pretende ser un prototipo cerrado, sino que, pretende plantear una propuesta de solución que sirva de base para que pueda ser extendido a cualquier proyecto y que cubra las necesidades del equipo encargado del desarrollo de modelos de Machine Learning.

Bibliografía

[1]

HeidiSteen, «About DeployR - DeployR 8.x» Disponible en: https://docs.microsoft.com/en-us/machine-learning-server/deployr/deployr-about.

[2]

«Azure Databricks | Microsoft Azure». Disponible en: https://azure.microsoft.com/es-es/services/databricks/.

[3]

«Cassandra Database Tutorial for Beginners: Learn in 3 Days». Disponible en: https://www.guru99.com/cassandra-tutorial.html.

[4]

«Cloud Machine Learning Engine Documentation | Cloud Machine Learning Engine (Cloud ML Engine)», *Google Cloud*. Disponible en: https://cloud.google.com/ml-engine/docs/.

[5]

«Compare Kubernetes vs Docker Swarm», *Platform9*. Disponible en: https://platform9.com/blog/kubernetes-docker-swarm-compared/.

[6]

J. Allen, *Creating APIs in R with Plumber*. Disponible en: https://www.rplumber.io/docs/.

[7]

«Dash User Guide and Documentation - Dash by Plotly». Disponible en: https://dash.plot.ly/.

[8]

«Documentation for Cassandra Storage Engine». Disponible en: http://cassandra.apache.org/doc/latest/architecture/storage_engine.html.

[9]

«Documentation for rpy2 — rpy2 2.8.4 documentation». Disponible en: https://rpy2.readthedocs.io/en/version_2.8.x/.

[10]

A. Zheng, «Evaluating Machine Learning Models», *O'Reilly Media*, 22-oct-2015. Disponible en: https://www.oreilly.com/ideas/evaluating-machine-learning-models.

[11]

«Get Docker», *Docker*, 05-mar-2018. Disponible en: https://www.docker.com/get-docker.

[12]

HeidiSteen, «Get started for Administrators configuring Machine Learning Server to operationalize». Disponible en: https://docs.microsoft.com/en-us/machine-learning-server/operationalize/configure-start-for-administrators.

[13]

«H2O.ai Documentation». Disponible en: http://docs.h2o.ai/.

[14]

HeidiSteen, «Install Machine Learning Server for Windows». Disponible en: https://docs.microsoft.com/en-us/machine-learning-server/install/machine-learning-server-windows-install.

[15]

«Introducing Py2PMML», *Help Desk*. Disponible en: http://support.zementis.com/hc/en-us/articles/231588567-Introducing-Py2PMML.

[16]

«Meet Michelangelo: Uber's Machine Learning Platform», *Uber Engineering Blog*, 05-sep-2017. Disponible en: https://eng.uber.com/michelangelo/.

[17]

«Modelos de aprendizaje automático y algoritmos | Amazon SageMaker en AWS». Disponible en: https://aws.amazon.com/es/sagemaker/.

[18]

«Omphalos, Uber's Parallel and Language-Extensible Time Series Backtesting Tool», *Uber Engineering Blog*, 24-ene-2018. Disponible en: https://eng.uber.com/omphalos/.

[19]

«OpenCPU - Producing and Reproducing Results»]. Disponible en: https://www.opencpu.org/.

[20]

HeidiSteen, «Operationalize models, analytics, & web services in Machine Learning Server». Disponible en: https://docs.microsoft.com/en-us/machine-learning-server/what-is-operationalization.

[21]

«PMML Export Functionality in R: Supported Packages», *Help Desk*. [En línea]. Disponible en: http://support.zementis.com/hc/en-us/articles/231587247-PMML-Export-Functionality-in-R-Supported-Packages. [Accedido: 04-jul-2018].

[22]

«Productos», *Tableau Software*. Disponible en: https://www.tableau.com/es-es/products.

[23]

reticulate: R Interface to Python. RStudio, 2018. Disponible en: https://github.com/rstudio/reticulate

[24]

«SAP Lumira | Visualización de datos | BI de autoservicio», *SAP*. Disponible en: https://www.sap.com/spain/products/lumira.html.

[25]

«Shiny». Disponible en: https://shiny.rstudio.com/.

[26]

HeidiSteen, «Supported Platforms for Machine Learning Server and Microsoft R Server». Disponible en: https://docs.microsoft.com/en-us/machine-learning-server/install/r-server-install-supported-platforms.

[27]

«Swarm mode overview», *Docker Documentation*, 28-jun-2018. Disponible en: https://docs.docker.com/engine/swarm/.

[28]

«Types of Performance Monitoring :: SAS(R) Decision Manager 2.2: User's Guide». Disponible en: http://support.sas.com/documentation/cdl/en/edmug/67015/HTML/default/viewer.htm#p1qz1z6cc486gkn1fxynb3upc7w0.htm.

[29]

«Welcome to Flask — Flask 1.0.2 documentation». Disponible en: http://flask.pocoo.org/docs/1.0/.

[30]

«What is a Container», *Docker*, 29-ene-2017. Disponible en: https://www.docker.com/what-container.

[31]

«What is Kubernetes? - Kubernetes». Disponible en: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[32]

W. Felter, A. Ferreira, R. Rajamony, y J. Rubio, «An updated performance comparison of virtual machines and Linux containers», 2015, pp. 171-172.

[33]

J. M. White, «*Bandit algorithms for website optimization*». Sebastopol, California: O'Reilly, 2013.

[34]

P. Chapman, T. Khabaza, y C. Shearer, «Step-by-step data mining guide», p. 76.

Anexo:1. Dockerfiles

En este apartado se muestra el código empleado para crear las imágenes empleadas que darán lugar a los diferentes microservicios utilizados lo largo del proyecto.

MySQL (Datalake)

```
# MySQL
FROM mysql:5.5

LABEL maintainer="maintainer@gmail.com"

# user: root
ENV MYSQL_ROOT_PASSWORD luis
```

Cassandra (BD Histórica y Live)

```
# Cassandra
FROM cassandra:2.1

LABEL maintainer="maintainer@gmail.com"
```

Template_r (Microservicio de desarrallo R)

```
# RStudio Server
FROM rocker/rstudio:3.4.0

LABEL maintainer="maintainer@gmail.com"

ENV USER luis
ENV PASSWORD luis

RUN apt-get update && apt-get install -y \
  libmariadb-client-lgpl-dev \
  default-jre \
  default-jdk \
  && R CMD javareconf
RUN apt-get install -y r-cran-rjava

COPY requirements.R /usr/local/src/requirements.R
COPY cassandra-jdbc-2.1.1 /usr/local/src/cassandra-jdbc-2.1.1

WORKDIR /usr/local/src

RUN Rscript requirements.R
```

Template_p (Microservicio de desarrollo Python)

```
# Python 3.6
FROM python:3.6-slim

LABEL maintainer="maintainer@gmail.com"

WORKDIR /usr/src/

COPY requirements.txt ./

RUN pip install --no-cache -r requirements.txt
```

Flask (Microservicio de producción Python)

```
# Flask
FROM python:3.6-slim

RUN mkdir -p /home/app

COPY . /home/app

WORKDIR /home/app

RUN pip install --no-cache -r requirements.txt

CMD ["python", "app.py"]
```

Plumber (Microservicio de producción R)

```
# Plumber
FROM trestletech/plumber

RUN apt-get update && apt-get install -y \
  libmariadb-client-lgpl-dev \
  default-jre \
  default-jdk \
  && R CMD javareconf
RUN apt-get install -y r-cran-rjava

RUN mkdir -p /home/app

COPY . home/app

WORKDIR home/app

RUN Rscript requirements.R

ENTRYPOINT []
CMD ["Rscript", "plumber.R"]
```

Dashboard

```
FROM python:3.6-slim

LABEL maintainer="maintainer@gmail.com"

RUN mkdir -p /home/app

COPY . /home/app

WORKDIR /home/app

RUN pip install --no-cache -r requirements.txt
RUN pip install plotly --upgrade

CMD [“gunicorn”, “-b”, “127.0.0.1:8888”, “app:server”]
```