Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Algoritmos de Feature Selection utilizados en estimación de esfuerzo de proyectos de desarrollo software

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Iván Iñaki Ajenjo Vicente

Tutores: Marta Fernández Diego, Fernando González Ladrón de Guevara

2019/2020

Resumen

Hoy en día hay poco trabajo de investigación centrado en las técnicas de *Feature Selection* (FS), incluidas las características categóricas y continuas en la literatura de Estimación del esfuerzo de desarrollo de *software*. Este documento aborda el problema de seleccionar las características más relevantes del conjunto de datos de ISBSG (International Software Benchmarking Standards Group) para su uso en la estimación de esfuerzo de desarrollo *software*. El objetivo es mostrar la utilidad de dividir en dos la lista clasificada de características proporcionadas por un enfoque secuencial de FS basado en información mutua, con respecto a características categóricas y continuas. Estas listas se recombinan posteriormente de acuerdo con la precisión de un modelo de razonamiento basado en casos. Por lo tanto, se comparan cuatro algoritmos de FS utilizando un conjunto de datos completo con 621 proyectos y 12 características de ISBSG. Por un lado, dos algoritmos solo consideran la relevancia, mientras que los dos restantes siguen el criterio de maximizar la relevancia y también minimizar la redundancia entre cualquier característica independiente y las características ya seleccionadas. Por otro lado, los algoritmos que no discriminan entre características continuas y categóricas consideran solo una lista, mientras que los que las diferencian utilizan dos listas que luego se combinan. Como resultado, los algoritmos que utilizan dos listas presentan un mejor rendimiento que los algoritmos que utilizan una sola lista. Por lo tanto, es significativo considerar dos listas diferentes de características para que las características categóricas se puedan seleccionar con mayor frecuencia.

Palabras clave: Feature Selection, Información mutua, desarrollo software, estimación de esfuerzo, ISBSG, k-nearest neighbor, Agile.

Abstract

There is still little research work focused on feature selection (FS) techniques including both categorical and continuous features in Software Development Effort Estimation (SDEE) literature. This paper addresses the problem of selecting the most relevant features from ISBSG (International Software Benchmarking Standards Group) dataset to be used in SDEE. The aim is to show the usefulness of splitting the ranked list of features provided by a mutual information-based sequential FS approach in two, regarding categorical and continuous features. These lists are later recombined according to the accuracy of a case-based reasoning model. Thus, four FS algorithms are compared using a complete dataset with 621 projects and 12 features from ISBSG. On the one hand, two algorithms just consider the relevance, while the remaining two follow the criterion of maximizing relevance and also minimizing redundancy between any independent feature and the already selected features. On the other hand, the algorithms that do not discriminate between continuous and categorical features consider just one list, whereas those that differentiate them use two lists that are later combined. As a result, the algorithms that use two lists present better performance than those algorithms that use one list. Thus, it is meaningful to consider two different lists of features so that the categorical features may be selected more frequently.

Keywords: Feature Selection, Mutual Information, Software development, effort estimation, ISBSG, k-nearest neighbor, Agile.

Este trabajo va dedicado a mi familia por apoyarme y ayudarme en todo momento para llegar hasta aquí.

Tabla de contenidos

Tabla de Figuras

Tabla de Códigos

1 Introducción

El Aprendizaje Automático consiste en una disciplina de las ciencias informáticas, relacionada con el desarrollo de la Inteligencia Artificial, y que sirve para crear sistemas que pueden aprender por sí solos.

Es una tecnología que permite hacer automáticas una serie de operaciones con el fin de reducir la necesidad de que intervengan los seres humanos. Esto puede suponer una gran ventaja a la hora de controlar una ingente cantidad de información de un modo mucho más efectivo.

Lo que se denomina aprendizaje consiste en la capacidad del sistema para identificar una gran serie de patrones complejos determinados por una gran cantidad de parámetros. Al fin y al cabo, la máquina no aprende por sí misma, sino un algoritmo de su programación que se modifica con la constante entrada de datos en la interfaz y de este modo puede predecir escenarios futuros o tomar decisiones de manera automática.

Los usos de esta tecnología son muy variados. Se utiliza para la detección de *malware*, en el comercio financiero, en el cuidado de la salud, el marketing personalizado, motores de búsqueda... Sus aplicaciones son muy variadas, pero en el ejemplo tratado en este trabajo la utilizaremos para predecir el esfuerzo de desarrollo de los proyectos software.

1.1 Motivación

Hoy en día se almacenan datos de forma masiva, en cualquier ámbito de la informática y en general de la tecnología. Estos son almacenados por si más adelante pueden ser de ayuda. Lo que esto nos produce es que tengamos una base de datos demasiado grande para poder trabajar con comodidad. Con esta gran cantidad de datos se pueden construir modelos muy útiles. Pero por otro lado estos complejos modelos suelen ser muy pesados lo cual encarece el coste de computación y por tanto el tiempo de análisis. Es por esto por lo que cada vez más se utilizan algoritmos de *Feature Selection* o de selección de variables. Estos algoritmos se encargan de seleccionar las mejores variables para un objetivo concreto, reduciendo así el peso de los modelos. Esto hace que sean más sencillos de interpretar, que el tiempo de entrenamiento sea más corto, reducir el sobreajuste, etc.

Por tanto, la premisa central al utilizarse una técnica de selección de variable es el hecho de que una base de datos contiene muchas variables redundantes o irrelevantes para un objetivo concreto y por tanto estas pueden ser eliminadas. Quiero recalcar la diferencia entre una variable irrelevante y, una redundante ya que esta diferencia es importante a la hora de plantear nuestros algoritmos. Las variables irrelevantes son aquellas que no contienen información útil para una variable objetivo. Mientras que una variable relevante, puede convertirse en redundante en presencia de otra variable con la que esté fuertemente relacionada.

En el desarrollo software, la estimación de esfuerzo es el proceso de predecir de la forma más realista posible la cantidad de esfuerzo requerido para desarrollar y mantener software. La práctica más común para la estimación de un proyecto software es la estimación de un experto. Una persona con experiencia, basándose en estas experiencias previas, realiza una estimación del esfuerzo del proyecto.

Utilizando aprendizaje automático esta estimación de esfuerzo puede verse mejorada. El aprendizaje automático como se ha comentado anteriormente se utiliza para muchos fines distintos, siendo uno de ellos los predictores. Y para la selección de variables para el modelo de predicción es de suma importancia contar con algoritmos de *Feature Selecction*.

1.2 Objetivos

El objetivo general del trabajo es el de analizar y evaluar distintos algoritmos de *Feature Selection* y utilizar las variables seleccionadas para la creación de modelos de predicción del esfuerzo de desarrollo software.

A continuación, se presentan los objetivos específicos:

- Se va a realizar el análisis de la precisión de los distintos algoritmos de FS, con el fin de determinar cuál es el algoritmo idóneo para los datos disponibles. Para analizar la precisión del algoritmo se van a utilizar los valores de MMRE.
- También se va a analizar los tiempos de ejecución de los distintos algoritmos de FS. Estos se van a comparar entre si para asegurar que algoritmo es el más eficiente en cuanto a tiempos de ejecución se refiere.
- Se va a realizar el análisis de las distintas variables seleccionadas por cada uno de los algoritmos. Al fin y al cabo, los algoritmos de FS se encargan de eso, de seleccionar las mejores variables para un objetivo concreto.
- Por último, vamos a realizar una comparativa entre dos versiones de la base de datos, la principal del trabajo con proyectos desarrollados con metodología tradicional y una ejecución con proyectos desarrollados utilizando metodologías ágiles. Se analizará el resultado de la ejecución en ambos casos y se comentarán las diferencias entre estos resultados.

1.3 Impacto esperado

Al fin y al cabo, estamos realizando un estudio sobre los algoritmos de *Feature Selection* y por tanto esperamos realizar un análisis de las variables que contiene la base de datos de ISBSG. Gracias a estos algoritmos generaremos el mejor modelo, seleccionando las mejores variables para nuestro predictor. Por lo tanto, obtendremos tanta información como sea posible de las variables disponibles.

El problema de la base de datos de ISBSG es la cantidad de variables categóricas que contiene. Además, algunas de estas variables no están correctamente formateadas y codificadas, lo cual complica el trabajo de analizarlas utilizando software. Precisamente este es el motivo de añadir este segundo tipo de algoritmos que utiliza dos listas. Esto se comentará en detalle más adelante pero básicamente estos se utilizan para equilibrar la selección de ambos tipos de variables.

1.4 Estructura

La memoria está estructurada en seis apartados más cinco anexos. La estructura de los apartados tiene el objetivo de organizar la información y el contenido con el fin de facilitar la lectura y ayudar a la comprensión del trabajo realizado.

- **Apartado 1, Introducción:** es el apartado actual. Donde se realiza una presentación del proyecto a realizar y se resumen algunos conceptos muy generales básicos para la comprensión del proyecto.
- **Apartado 2, Estado del arte:** en este apartado se explica el estado actual de los distintos conceptos clave para comprender el funcionamiento del script y la motivación de este.
- **Apartado 3, Algoritmos de Feature Selection basados en MI:** en este apartado se realiza la explicación de los distintos algoritmos utilizados en la comparación de este trabajo. También se explican las diferencias entre ellos.
- **Apartado 4, Metodología:** en este apartado se explica en detalle el funcionamiento del script y como se utilizan las herramientas al alcance para analizar los distintos resultados.
- **Apartado 5, Resultados experimentales:** en este apartado se realiza el análisis de los resultados obtenidos por el algoritmo del trabajo. También se realizan una serie de gráficas para facilitar la comprensión de estos.
- **Apartado 6, Conclusiones:** en este apartado se realiza el análisis del proyecto a nivel global, así como una pequeña introducción a cuál sería el trabajo futuro y las posibles mejoras para realizar.
- **Anexo:** en este apartado se incluyen tanto capturas de algún código alternativo, como gráficas de resultados de las distintas librerías que se han probado. También se incluyen algunas anotaciones y conclusiones sobre las herramientas utilizadas.

2 Estado del arte

2.1 ISBSG[1]

ISBSG (*International Software Benchmarking Standards Group*) es una organización sin ánimo de lucro, fundada en Australia en 1997 por un grupo de asociaciones de métricas de software. El objetivo de la organización es el de promover la utilización de datos en la industria IT para mejorar el proceso y los productos software.

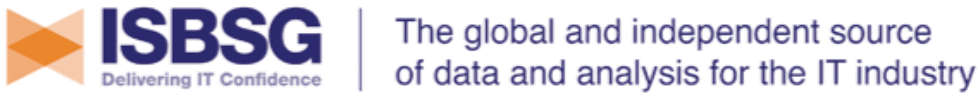

Figura 1 Logo de ISBSG, fuente: https://www.isbsg.org/ [I]

Hoy en día ISBSG cuenta con la colaboración de una gran variedad de asociaciones de métricas y corporaciones privadas. Clientes por todo el mundo utilizan los datos de la organización, los informes y las herramientas de estimación para mejorar la planificación de proyectos software.

[1] https://www.isbsg.org/

ISBSG diseñó y mantiene dos repositorios internacionales [I] públicos para mejorar la gestión de recursos de IT para negocios y gobiernos. El conjunto de datos de ISBSG ofrece una gran cantidad de información sobre proyectos de software finalizados, que permite el *benchmarking*, monitoreo, control de calidad… Sin embargo, hay cuestiones que deben tenerse en cuenta a la hora de utilizar estos repositorios (Fernández-Diego y González Ladrón-de-Guevara 2014 [II]). El trabajo experimental de este documento se basa en *ISBSG Release 12* que incluye 6006 proyectos y 126 características.

2.2 Estimación de esfuerzo

Como se ha comentado anteriormente la estimación de esfuerzo es un proceso mediante el cual se predice o estima un esfuerzo para un proyecto concreto. Esta estimación se lleva a cabo en prácticamente todo tipo de industrias. Y realmente al igual que en el esfuerzo de desarrollo software en una inmensa mayoría de casos esta estimación se hace basándose en experiencias previas [III]. Por tanto, podríamos decir que la estimación la realiza un experto. No se debe confundir la estimación de esfuerzo con la estimación de tiempo. El esfuerzo se refiere a la suma de los tiempos que dedicarán los diferentes recursos al proyecto.

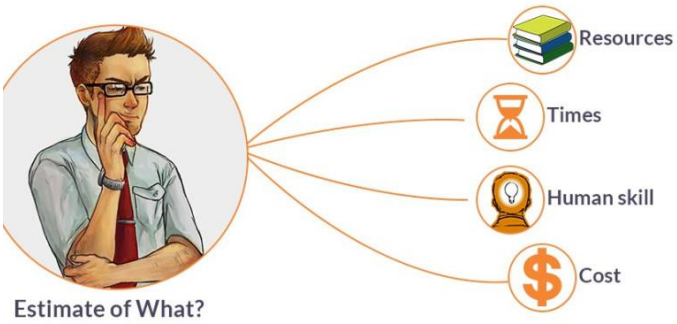

Figura 2 Tipos de estimaciones, Fuente: renierbotha.com

En cambio, cuando hablamos de la estimación de tiempo, nos referimos al periodo en el calendario que será necesario para poder cumplir ciertos objetivos. Por ejemplo, podemos determinar un tiempo necesario de tres meses para completar un proyecto, mientras que el esfuerzo varía en función de la cantidad de personas que trabajen en dicho proyecto.

Normalmente las estimaciones de esfuerzo son demasiado optimistas y tienen demasiada confianza en su propia estimación. Por tanto, en muchos de los casos las estimaciones son inferiores al esfuerzo real y esta tendencia no parece haber cambiado en los últimos años [IV]. De hecho, a causa de esta tendencia se han creado ciertas citas con un toque de humor relacionadas con la subestimación de esfuerzo, como la de Tom Cargill conocida como:

Ninety-ninety rule:

The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.

Esto no solo ocurre en la industria de la informática. En el resto de las industrias también es muy difícil realizar una estimación tanto de esfuerzo como de tiempo. Al igual que en

el desarrollo software la mayor parte de las estimaciones se realizan basándose en la experiencia previa de un experto. Sin embargo, en la mayoría las semejanzas entre los proyectos hacen que sea más fácil realizar estas estimaciones. Esto en el desarrollo software no tiene porqué ser así ya que el mismo equipo de desarrollo que realiza un producto software puede realizar uno totalmente distinto. Por tanto, su experiencia previa no es tan valiosa.

Esta situación se repite en las empresas que trabajan bajo pedido. Es decir que el producto que desarrollan se hace íntegramente bajo pedido y personalizado para los clientes. Por ejemplo, las industrias navales o de maquinaria pesada, etc.

2.3 Feature Selection

En *Machine Learning* y estadística, el proceso de seleccionar un subconjunto de características pertinentes para su uso en construcción de modelos se conoce como *Feature Selection* (FS). [V]

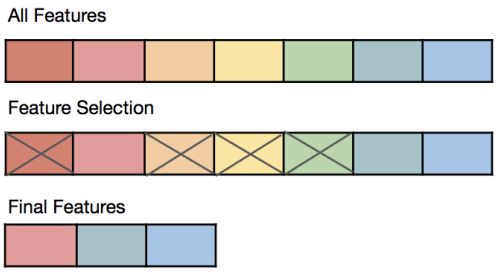

Figura 3 Proceso de Feature Selection, fuente: kdnuggets.com

El objetivo de utilizar técnicas de FS es el de reducir el conjunto de variables con el que se trabaja a aquellas variables que son más relevantes, como se muestra en la Figura 3, de forma que se eliminan características redundantes e irrelevantes. Las características redundantes e irrelevantes son dos tipos distintos, ya que una característica relevante puede ser redundante en presencia de otra característica relevante.

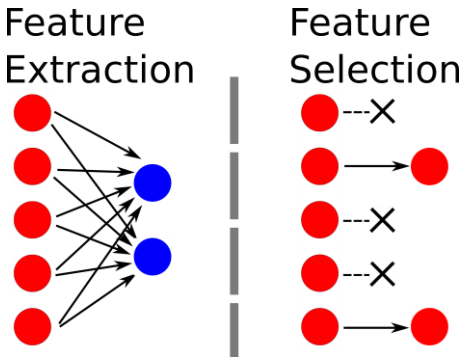

Figura 4 Feature Selection y Feature Extraction, fuente: groundai.com

No se debe confundir con *Feature Extraction*, método a partir del cual se generan nuevas variables o características a partir de las existentes con *Feature Selection*. La figura 4 pone de manifiesto esta diferencia.

Un algoritmo de FS puede ser visto como una combinación de una técnica de búsqueda para proponer nuevos subconjuntos de variables, junto con un evaluador que mide y puntúa los distintos subconjuntos. La elección del evaluador influye fuertemente en el algoritmo. Y son estas evaluaciones las que distinguen entre las tres categorías principales de algoritmos de selección de variables.

- Envolvedores (*Wrappers*)
- Filtrado
- Embebidos

Como se ha comentado anteriormente FS es un proceso muy importante, que reduce el coste de computación y el "ruido". Es decir, todas las variables que en realidad no nos aportan nada se descartarían utilizando FS. Por tanto, utilizando estas técnicas y combinándolas con algunas otras obtenemos modelos cada vez de mayor calidad.

2.4 Mutual Information

En la ciencia de datos hay muchos tipos de variables, en la propia base de datos de ISBSG hay unas 120 diferentes. Cada una tiene sus características propias, aunque las hay muy similares. Por ejemplo, si quisiéramos crear una base de datos de personas podríamos almacenar distintos datos, como por ejemplo su nombre, su edad, su estatura y peso etc. De algunas de estas variables no podríamos obtener información entre ellas. Por ejemplo, si comparamos el nombre de una persona, con su dirección sería prácticamente imposible averiguar el valor de una a partir del valor de la otra. Pero si por el contrario comparamos su género con su altura sí que podríamos llegar a una predicción posible, ya que como norma general los hombres son más altos que las mujeres. Evidentemente podemos encontrar estas variables con información mutua y podemos utilizarlas entonces a la hora de realizar predicciones.

La *mutual information* o información mutua entre dos variables aleatorias es una cantidad que mide la dependencia mutua de las dos variables, es decir, mide la reducción de la incertidumbre de una variable aleatoria, X, debido al conocimiento de otra variable aleatoria, Y. [VI]

En este trabajo, los algoritmos de selección de variable implementados y analizados hacen uso del concepto Teoría de la Información.

3 Algoritmos de FS propuestos basados en MI

Los algoritmos de FS siguen el siguiente proceso. Se parte de una base de datos en la que tenemos definida la variable objetivo. Al modelo de predicción le añadimos variables de una en una. Cada modelo se evalúa y si este mejora el modelo anterior, proseguimos con la inclusión de una nueva variable. El proceso termina cuando no nos quedan variables por añadir, es decir todas han sido evaluadas.

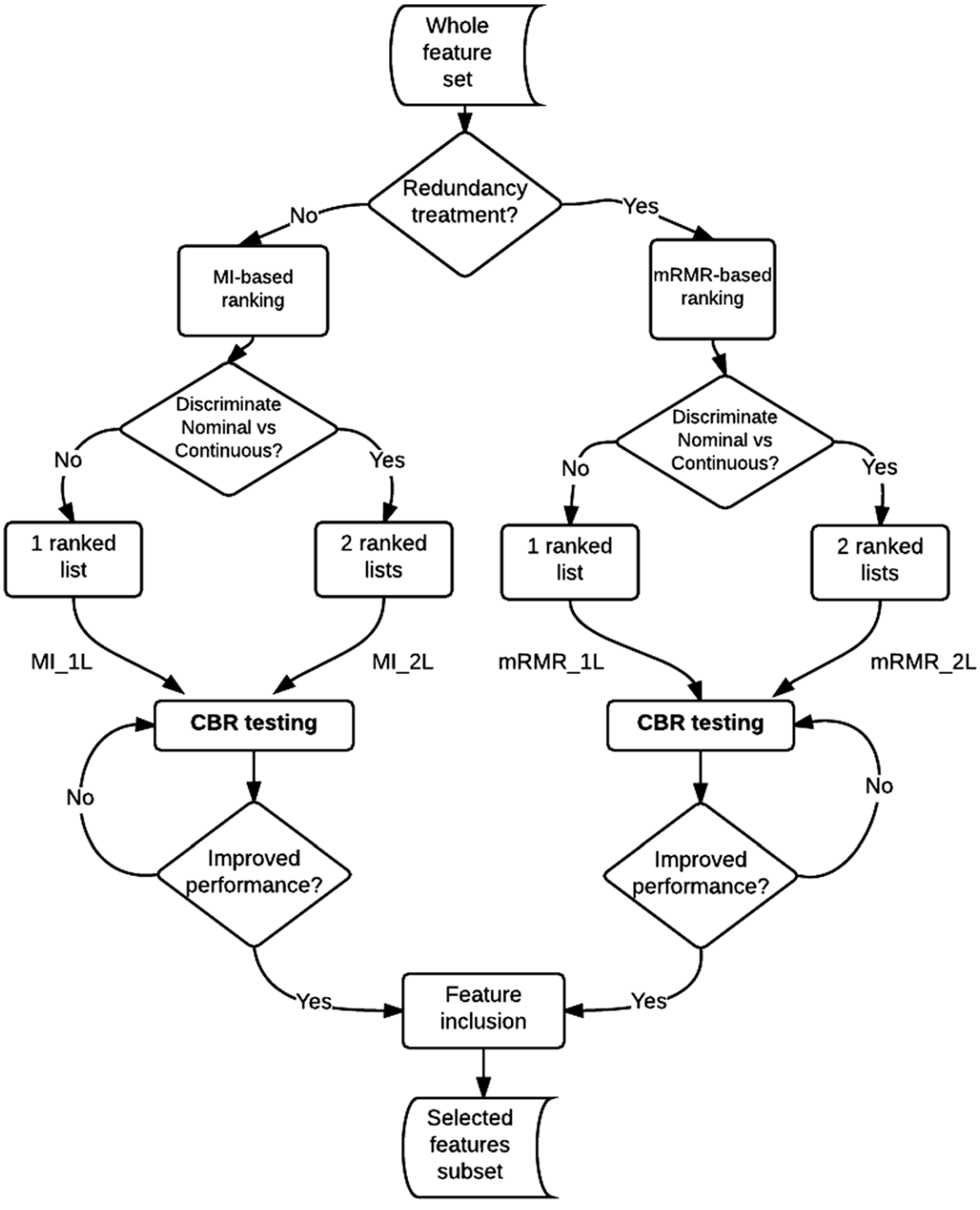

Figura 5 Diagrama de bloques de los algoritmos de FS propuestos.

3.1 MI vs MRMR

El algoritmo de MI ordena las variables en función de la información mutua respecto a nuestra variable objetivo.

- MI. Este algoritmo emplea una estrategia sencilla. Consiste en añadir en cada paso la mejor variable de acuerdo con un criterio específico. Las variables se ordenan de acuerdo con la relevancia respecto a la variable objetivo o dependiente utilizando MI. Esta clasificación es obtenida utilizando el *dataset* completo. Después todas las variables son probadas de forma secuencial para decidir su inclusión o no en el modelo de predicción. Para saber si la variable es incluida en el modelo de predicción se prueba si mejora el modelo CBR previo en términos de MMRE. En caso de ser así la variable debe ser incluida entre las variables elegidas y en caso contrario se descarta. [I]

El algoritmo mRMR (Minimum Redundancy Maximum Relevance) selecciona las variables con más información mutua con respecto a la variable obetivo reduciendo la redundancia entre estas.

- mRMR. Se puede decir que una variable muy relevante para la dependiente puede ser inútil en caso de que su información se pueda obtener de otra de las variables seleccionadas. En ese caso la variable no debe ser seleccionada. Para resolver esto en cada paso de la búsqueda podemos seleccionar la variable con más diferencia entre la relevancia y la redundancia con las variables seleccionadas. La única diferencia con el algoritmo de MI_1L es que en este caso las variables se ordenan de acuerdo con el criterio de mRMR, mínima redundancia máxima relevancia. [I]

3.2 1 lista vs 2 listas

Greedy forward selection y doquire forward selection son las funciones que ejecutaran los algoritmos de 1L y de 2L respectivamente para la selección de variables. En función del tipo de algoritmo se le pasa a la función la lista de variables ordenada por MI o por mRMR.

```python
def greedy_forward_selection(valor_knn, variable, var_ordenadas, df, umbral_mmre=0, verbose=False):
    valor_de_nfolds = 3
    total_iteraciones = len(var_ordenadas)
    umbral = 1 + umbral_mmre/100
    variables_elegidas = []
    variables_eliminadas = []
    mmres = []
    mmre_min = float('Inf')
    iteracion = 1
    while iteracion <= total_iteraciones:
        campos = [variable] + variables_elegidas + [var_ordenadas[0]]
        mmre_calc = evaluator(valor_de_nfolds, valor_knn, df[campos], variable)
        if ((umbral*mmre_min) >= mmre_calc):
            variables_elegidas.append(var_ordenadas[0])
            mmres.append(mmre_calc)
            if mmre_min > mmre_calc:
                mmre_min = mmre_calc
        else:
            variables_eliminadas.append(var_ordenadas[0])
        var_ordenadas.pop(0)

        if verbose:
            print('Iteracion', iteracion, 'de', total_iteraciones)
            print('Variables elegidas', variables_elegidas)
            print('Variables eliminadas', variables_eliminadas)
        iteracion += 1
    resultado = [variable, variables_elegidas,
                 variables_eliminadas, mmres, umbral_mmre]
    return resultado
```

Código 1 GFS

En el Código 1 se muestra la implementación del algoritmo GFS. En el caso del DFS, la implementación se muestra en el Código 2, es algo más compleja ya que se emplean 2 listas 1 con las variables numéricas y otra con las variables categóricas. Como se puede ver en la iteración del bucle en DFS, la diferencia está en el hecho de trabajar con 2 listas de variables seleccionando así la que produce una mejoría mayor en el MMRE.

```
while hay_nominales or hay_numericas:
    mmre_num = float('Inf')
    mmre_nom = float('Inf')
    if len(var_numericas) > 0:
        campos = [variable] + variables_elegidas + [var_numericas[0]]
        mmre_num = evaluator_r(
            valor_de_nfolds, valor_knn, df[campos], variable)

    if len(var_nominales) > 0:
        campos = [variable] + variables_elegidas + [var_nominales[0]]
        mmre_nom = evaluator_r(
            valor_de_nfolds, valor_knn, df[campos], variable)

    if mmre_num <= mmre_nom:
        if umbral*mmre_min >= mmre_num:
            variables_elegidas.append(var_numericas[0])
            mmres.append(mmre_num)
            if mmre_min > mmre_num:
                mmre_min = mmre_num
        else:
            variables_eliminadas.append(var_numericas[0])
        var_numericas.pop(0)
    else:
        if umbral*mmre_min >= mmre_nom:
            variables_elegidas.append(var_nominales[0])
            mmres.append(mmre_nom)
            if mmre_min > mmre_nom:
                mmre_min = mmre_nom
        else:
            variables_eliminadas.append(var_nominales[0])
        var_nominales.pop(0)
```

Código 2 DFS

4 Metodología

4.1 Herramientas

4.1.1 Lenguajes de programación

4.1.1.1 Python

Python[2] es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad del código. Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y en menor medida programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License.

Se añade más información de utilidad sobre Python en los anexos.

En el contexto de este trabajo la decisión estaba entre utilizar R o Python. Finalmente se decidió hacerlo en Python por la experiencia previa con el lenguaje.

4.1.1.2 R

R[3] es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

R nació como una reimplementación de software libre del lenguaje S, adicionado con soporte para alcance estático. Se trata de uno de los lenguajes de programación más utilizados en investigación científica, siendo además muy popular en los campos de aprendizaje automático, minería de datos, investigación biomédica, bioinformática y matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes librerías o paquetes con funcionalidades de cálculo y *graficación*.

Pese a que no es el lenguaje principal de programación del proyecto ha sido una herramienta muy útil y utilizada durante el desarrollo de este.

[2] https://www.python.org/
[3] https://www.r-project.org/

4.1.2 Entornos

4.1.2.1 Anaconda

Anaconda[4] es una distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos, y aprendizaje automático. Esto incluye procesamiento de grandes volúmenes de información, análisis predictivo y cómputos científicos. Está orientado a simplificar el despliegue y administración de los paquetes de software.

Las diferentes versiones de los paquetes se administran mediante el sistema de gestión de paquetes conda, el cual hace bastante sencillo de instalar, correr, y actualizar software de ciencia de datos y aprendizaje automático.

4.1.2.2 Jupyter Notebook

Jupyter[5] Notebook (anteriormente IPython Notebooks) es un entorno informático interactivo, *open source*, basado en la web para crear documentos. El término "notebook" puede hacer referencia coloquialmente a muchas entidades diferentes, principalmente la aplicación web Jupyter, el servidor web Jupyter Python o el formato de documento Jupyter según el contexto. Un documento de Jupyter Notebook es un documento JSON, que sigue un esquema versionado y que contiene una lista ordenada de celdas de entrada/salida que pueden contener código, texto (usando Markdown), fórmulas matemáticas, gráficos y texto enriquecidos, generalmente terminado con la extensión ".ipynb".

Jupyter Notebook se puede convertir a varios formatos de salida estándar (HTML, PDF ...)

En lugar de utilizar Jupyter Notebook directamente sobre el navegador web, se utiliza su integración completa con Visual Studio Code.

4.1.3 Librerias

4.1.3.1 Pandas

En computación y ciencia de datos Pandas[6] es una biblioteca de software escrita como extensión de NunPy para manipulación y análisis de datos para el lenguaje de programación Python. Ofrece estructuras de datos y operaciones para manipular tablas numéricas.

[4] https://www.anaconda.com/
[5] https://jupyter.org/
[6] https://pandas.pydata.org/

Pandas es una librería para el análisis de datos que cuenta con las estructuras necesarias para limpiar los datos en bruto y que sean aptos para el análisis. Pandas es capaz de realizar tareas importantes como, fusionar datos o el tratamiento de datos perdidos.

La estructura básica de datos de Pandas es el *Dataframe*, una colección ordenada de columnas con nombres y tipos, parecido a una tabla de una base de datos. Sobre este se pueden aplicar filtros o realizar consultas para obtener la información deseada.

4.1.3.2 Scikit-Learn

Scikit-learn [7] es una librería para aprendizaje automático de software libre para el lenguaje de programación Python. Incluye algoritmos de clasificación, regresión y análisis de grupos, k-means, etc. Está diseñada para interoperar con librerías numéricas y científicas como NumPy.

La gran variedad de algoritmos y utilidades de scikit-learn la convierten en una herramienta básica. En nuestro caso se utilizará tanto para calcular la Mutual Information como para realizar los K-fold.

4.1.3.3 Matplotlib

Matplotlib[8] es una librería para la generación de gráficos a partir de datos contenidos en listas o arrays en Python. Proporciona una Api, pylab, diseñada para recordar a la de MATLAB. Se ha utilizado para generar todos los gráficos que aparecen a lo largo del trabajo.

4.1.3.4 Seaborn

Seaborn [9] es una librería para hacer gráficos estadísticos en Python. Está construida sobre matplot y tiene una integración muy desarrollada con las estructuras de datos de Pandas. Será la herramienta utilizada para generar gran parte de los gráficos de resultados. Genera gráficos más atractivos e informativos de una forma sencilla.

4.2 Pre-procesado de los datos

4.2.1 Filtrado

Ya que ISBSG es un *dataset* muy grande y heterogéneo, es necesario un proceso de preparación de datos antes de cualquier análisis.

Como paso inicial se realiza la importación de la BD. En este caso se trata de un archivo csv delimitado por punto y coma. Como el archivo contenía más información que la propia base de datos, todos los datos innecesarios tales como recordatorios legales o de

[7] https://scikit-learn.org/
[8] https://matplotlib.org/
[9] https://seaborn.pydata.org/

categorización de variables se eliminan. Al ser solamente es necesario hacerlo 1 vez este proceso se realiza manualmente. Se accede al archivo y se eliminan las 5 primeras líneas.

A partir de ahí, para realizar la importación de la BD se utiliza la librería Pandas la cual genera automáticamente un *Dataframe* con los datos importados, que requieren una serie de operaciones de limpieza. Se opta por realizar el trabajo utilizando Jupyter Notebook. Esto nos permite comprobar en todo momento el estado de las distintas variables que se van generando, pudiendo ver los cambios en tiempo real sin necesidad de ejecutar todo el script cada vez que se quiere cambiar algo de código.

```
archivo = '..\data\ISBSG DATA Release 12.csv'
df = pd.read_csv(archivo, sep = ';', low_memory = False)
```

Código 3 Importación de la BD

Con el archivo correctamente formateado la importación se realiza en 1 línea de código, como se muestra en el código 3.

Para seleccionar un primer grupo de columnas o features se puede utilizar la función loc()[10] pasándole una lista con los identificadores de las columnas a seleccionar. Esto aparece a continuación…

```
variables = ['Data Quality Rating', 'UFP rating', 'Industry Sector','Application Group',
'Development Type', 'Development Platform', 'Language Type', 'Primary Programming Language',
'Count Approach', 'Functional Size', 'Adjusted Function Points', 'Normalised Work Effort Level
1', 'Summary Work Effort', 'Project Elapsed Time', 'Business Area Type', '1st Data Base
System', 'Used Methodology', 'Resource Level', 'Max Team Size', 'Average Team Size', 'Input
count', 'Output count', 'Enquiry count', 'File count', 'Interface count']
df = df.loc[:, variables]
```

Código 4 Selección de variables

De esta forma vamos acotando el *dataset* con las columnas que queremos.

A continuación, trabajamos por filas. La Tabla 2 resume los criterios de selección de proyectos, que se especifican a continuación:

Tabla 1 Criterios de selección de proyectos

Criterio de selección	Proyectos Restantes
Calidad de datos general alta	
Calidad funcional alta	3935
Esfuerzo del equipo de desarrollo conocido	
Esfuerzo del ciclo de vida completo	2249
IFPUG versión 4.0+	1884

- Calidad de datos general alta

df['Data Quality Rating'] == 'A') | (df['Data Quality Rating'] == 'B'

- Calidad functional alta

[10] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.*Dataframe*.loc.html

df['UFP rating'] == 'A') | (df['UFP rating'] == 'B'

- Esfuerzo del equipo de desarrollo conocido

df['Normalised Work Effort Level 1'].notnull()

- Esfuerzo del ciclo de vida completo

df['Normalised Work Effort Level 1'] == df['Summary Work Effort']

- IFPUG version 4.0+

df['Count Approach'] == 'IFPUG 4+'

```
filtro = ((df['Data Quality Rating'] == 'A') | (df['Data Quality Rating'] == 'B')) & ((df['UFP rating']
== 'A') | (df['UFP rating'] == 'B'))
df = df.loc[filtro, :]
filtro = (df['Normalised Work Effort Level 1'].notnull()) & (df['Normalised Work Effort Level 1'] == df
['Summary Work Effort'])
df = df.loc[filtro, :]
filtro = df['Count Approach'] == 'IFPUG 4+'
df = df.loc[filtro, :]
```

Código 5 Filtrado de proyectos

Como se puede observar en el fragmento de código (Código 4), para realizar los distintos filtrados utilizamos de nuevo la función loc(). Pero en esta introducimos las condiciones lógicas que deben superar nuestros proyectos objetivo.

Esta operación no se ha realizado solamente con esta configuración de filtrado, se han planteado distintas configuraciones con el fin de realizar análisis más exhaustivos y obtener una mayor cantidad de información. De hecho, un análisis similar se ha realizado con la versión 2017 R1 de la base de datos de ISBSG, con el fin de obtener un subset de proyectos ágiles óptimo para la realización del trabajo. Como no se pudo obtener una configuración óptima esto se descartó y finalmente se utilizó la comentada anteriormente.

4.2.2 Set inicial de features

Tres variables de esfuerzo están disponibles en el *dataset* de ISBSG. La fundamental es Summary Work Effort (SWE), medido en horas. Es el esfuerzo total del proyecto contribuido por las empresas colaboradoras, pero SWE no cubre todas las fases del ciclo de vida del proyecto. Normalised Effort es la estimación de ISBSG del esfuerzo total cuando alguna de las fases que faltan son añadidas. Aun así, puede haber algunas inconsistencias entre proyectos, incluso cuando se utiliza Normalised Effort, porque el reporte de este esfuerzo proviene de diferentes participantes y esto se indica en la variable Resource Level. Level 1 implica a que el esfuerzo es reportado solamente por el equipo de desarrollo. Los Level 2 y 3 añaden el esfuerzo del equipo de soporte y las operaciones computacionales y el Level 4 añade el esfuerzo de los usuarios finales y los clientes. Por tanto, Normalised Work Effort Level 1 es el esfuerzo normalizado del equipo de desarrollo solamente.

Para empezar, nos quedaremos con 20 de las variables independientes más utilizadas en la estimación de modelos de esfuerzo según Fernandez Diego y Gonzalez Ladrón de Guevara [VII]: Data Quality Rating, UFP rating, Industry Sector, Application Group,

Development Type, Development Platform, Language Type, Primary Programming Language, Count Approach, Functional Size, Adjusted Function Points, Normalised Work Effort Level 1, Summary Work Effort, Project Elapsed Time, Business Area Type, 1st Data Base System, Used Methodology, Resource Level, Max Team Size, Average Team Size.

De este set inicial de 20 descartaremos variables con un nivel de datos perdidos superior al 60%: Average Team Size, Business Area Type, Max Team Size e Input Count, Output Count, Enquirity Count, File Count e Interface Count.

También nos aseguraremos de que NWEL1 no tiene valores nulos y que los valores de Resource Level sean 1. Después de esto Resource Level puede ser descartada del set de variables puesto que ya no nos aportará información relevante.

En este momento el subset incluye 1884 proyectos, 11 variables independientes y la dependiente NWEL1.

Por último, nos deshacemos de todos los proyectos que tienen valores nulos en alguna de las variables seleccionadas, lo que nos da un *dataset* final de 621 proyectos y 12 variables. Las variables independientes son las siguientes:

- Adjusted Function Points (AFP) es el tamaño ajustado para IFPUG, NESMA, FiAMA y MARK II. El tamaño es ajustado por un factor de conversión a AFP.

- Aplication Group (AG) es una variable derivada que agrupa Application Type de los proyectos en un único valor.

- 1st Data Base System (1DBS), la base de datos primaria utilizada en el proyecto. Esta variable tendrá que ser tratada más adelante en la categorización.

- Development Platform (DP) define la Plataforma de desarrollo determinada por el sistema operativo utilizado. Cada proyecto está clasificado como PC, Mid Range, Mainframe o Multi-Platform. DP es el mejor indicador del entorno en el que un proyecto es desarrollado.

- Development Type (DT) define si el Proyecto es un New Development, Enchancement o Re-Development

- Functional Size (FSZ) representa una función no ajustada de tamaño.

- Industry Sector (IS) identifica el tipo de organización que cede los datos del proyecto

- Language Type (LT) define el tipo de lenguaje de programación utilizado para el proyecto. La tercera generación es la dominante en nuestro subset, seguido de los de cuarta generación. En la práctica los lenguajes de 4a generación requieres un esfuerzo menor en la fase de programación, pero requieren un esfuerzo mayor en la fase de diseño.

- Project Elapsed Time (PET) representa el total de tiempo que ha transcurrido para el proyecto en meses.

- Primary Programming Language (PPL) indica cual es el lenguaje de programación principal del proyecto. Como los lenguajes de programación son de un tipo u otro en concreto esta información suele ser redundante con LT.

- Used Methodology (UM) define cuando una metodología ha sido utilizada en el desarrollo de un proyecto o no.

La Tabla 3 recoge las variables seleccionadas tras el proceso de filtrado especificando la naturaleza de las mismas: continua o categórica, característica fundamental en el presente trabajo:

Tabla 2 Variables seleccionadas de ISBS

Adjusted Function Points	AFP	Continua
Aplication Group	AG	Categórica
1st Data Base System	1DBS	Categórica
Development Platform	DP	Categórica
Development Type	DT	Categórica
Functional Size	FSZ	Continua
Industry Sector	IS	Categórica
Language Type	LT	Categórica
Project Elapsed Time	PET	Continua
Primary Programming Language	PPL	Categórica
Used Methodology	UM	Categórica
Normalised Work Effort Level 1	NWEL1	Continua

4.2.3 Categorización

Algunas de estas variables tienen demasiados valores distintos o no están codificadas adecuadamente. Por tanto, se deben normalizar para minimizar la confusión y maximizar la consistencia.

En concreto se han recategorizado 2 variables: PPL y 1DBS. En el caso de PPL 2 de los proyectos tenían valores inválidos los cuales se han codificado como “Unspecified” y otros 3 proyectos se han incluido en categorías más comunes. Con estos cambios se han obtenido 32 valores diferentes.

En el caso de 1DBS ha sido algo más complejo. Como se ha comentado anteriormente 1DBS es la tecnología de base de datos primaria del software. Esta variable no está normalizada y simplemente incluye cadenas descriptivas en lugar de categorías predefinidas en un formato consistente. Algunos de los valores como “Yes”, “Multiple”, “ISAM”, etc, se han codificado como “Unspecified”. Y en el resto de los valores se han agrupado, por ejemplo “Oracle 7” y “Oracle 7.3”, se han codificado simplemente como “Oracle”.

Por otro lado, para la mayoría de los proyectos aparece más de 1 sistema, o está repleto de “;”, o un mismo valor aparece codificado de múltiples formas.

Es por ello necesario realizar una serie de operaciones para limpiar los valores de esta columna. En este caso utilizaremos un diccionario y expresiones regulares [11]. Las

[11] https://www.w3schools.com/python/python_regex.asp

expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. Proporcionan una manera muy flexible de buscar o reconocer cadenas de texto.

```
df['Primary Programming Language'].replace( programmingLenguaje, inplace = True)
database = {'[;].*':';','ACCESS[; ].*':'ACCESS', 'MS Access':'ACCESS', 'ACCESS;':'ACCESS',
'ADABAS;':'ADABAS', 'Micosoft.*':'Attain', 'DB2[; /].*':'DB2', 'IBM DB2':'DB2', 'UDB2':'DB2',
'Domino[ ].*':'Domino', 'LOTUS.*':'Domino', 'Notes.*':'Domino', 'Exchange.*':'Exchange',
'FOXPRO;':'Foxpro', 'HIRDB;':'HIRDB', 'DB[/].*':'IMS', 'DEDB;':'IMS', 'IDMS[; -].*':'IMS',
'IMS.*':'IMS', 'MS[- ]SQL[; ].*':'MS SQL', 'MSDE.*':'MS SQL', 'SQL Server[; ].*':'MS SQL',
'SQL;':'MS SQL', 'VSE/.*':'MS SQL', 'NCR;':'NCR', 'Oracle.*':'ORACLE', 'Personal O.*':'ORACLE',
 'RDB[; ].*':'ORACLE', 'CICS;':'ORACLE', 'SAS;':'SAS', 'Solid;':'Solid', 'SYBASE.*':'SYBASE',
'YES':'Unspecified', 'ISAM;':'Unspecified', 'multiple;':'Unspecified', 'VSAM[; ]
.*':'Unspecified', 'WATCOM[; ].*':'Watcom', 'WGRES;':'WGRES'}
df['1st Data Base System'].replace( database, inplace = True, regex = True)
df['1st Data Base System'].replace( {'ACCESS;':'ACCESS'}, inplace = True, regex = True)
```

Código 6 Recodificación de 1DBS

Como puede verse en el Código 5 hay que definir cada uno de los valores deseados para cada una de las expresiones regulares, las cuales se encargarán de realizar la búsqueda de los patrones y remplazar el valor por el definido.

Para comprobar nuestros resultados es muy útil la utilización de la función value_counts()[12], la cual nos devuelve en orden descendente las variables ordenadas por la cantidad de veces que aparecen. Podemos observar rápidamente si hay alguna variable que no cumple ningún requisito anterior y por tanto se recodifica.

4.3 Cross-validation múltiple [VIII] [IX]

La validación cruzada o cross-validation es una técnica muy utilizada para evaluar los resultados de un modelo estadístico. Proviene de la mejora del método de retención. Consiste en dividir en varios conjuntos complementarios los datos disponibles, generar el modelo con un subconjunto y validarlo con el otro subconjunto.

En la validación cruzada de K iteraciones los datos se dividen en K subconjuntos. Uno de los subconjuntos se utiliza para generar el modelo y el resto (k-1) se utilizan como datos de entrenamiento. El proceso es repetido durante k iteraciones, utilizando una vez cada uno de los subconjuntos para la generación de modelos. Finalmente se hace la media aritmética de los resultados de cada iteración para obtener un único resultado. Se trata de un método más preciso ya que evaluamos a partir de K combinaciones de datos de entrenamiento. Sin embargo, existe una clara desventaja, su coste desde un punto de vista computacional es muy elevado. En este trabajo, además, validación cruzada se ejecuta 500 veces. Por eso hablamos de validación cruzada múltiple. En conclusión, para cada ejecución todo el data-set, los 621 proyectos se utilizan, pero cada vez las particiones son diferentes ya que las k=3 divisiones se obtienen de forma aleatoria. No se ha utilizado ningún tipo de *seed* con el fin de que estas particiones fueran totalmente aleatorias en las distintas ejecuciones del script.

[12] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html

4.3.1 Algoritmo MMRE

El cálculo del MMRE se ha utilizado KNNImputer[13] para realizar la imputación del valor a estimar utilizando los k vecinos más cercanos. El algoritmo de KNN [X] selecciona los k vecinos más similares, es decir, los proyectos que más se parecen para realizar una estimación del proyecto objetivo. Para esta solución se crea un *dataframe* de resultados, donde tenemos una columna con el valor original y otra con el valor estimado. Por lo tanto, el algoritmo guarda el valor original, lo elimina del *dataframe* y lo imputa. Una vez imputado se recoge el valor para guardarlo en nuestro *dataframe* de resultados y el proceso se repite con la siguiente fila del *dataframe* original.

$$\frac{\sum |Valor\ Original - Valor\ Imputado| / Valor\ Original}{Total\ de\ Proyectos}$$

```
total = len(df)
resultado = pd.DataFrame(columns=['Valor Original', 'Valor Imputado'])
numero_columna = df.columns.get_loc(variable)
for i in range(total):
    df_test = df.copy(deep=True)
    dato_original = df_test[variable].iloc[i]
    df_test[variable].iloc[i] = np.nan
    imputer = KNNImputer(n_neighbors=k)
    df_test = imputer.fit_transform(df_test)
    dato_imputado = df_test[i, numero_columna]
    resultado = resultado.append(
        {'Valor Original': dato_original, 'Valor Imputado': dato_imputado}, ignore_index=True)
    mmre = (1/total)*sum(abs(resultado['Valor Original'] -
                             resultado['Valor Imputado'])/resultado['Valor Original'])
return mmre
```

Código 7 Calculo de MMRE

En la funcion final implementada que se muestra en el Código 8 solamente se devuelve el valor de MMRE para el *dataframe* determinado. Sin embargo, durante la fase de desarrollo se devuelve también el *dataframe* con los valores imputados. Esto es de gran ayuda para el proceso de *debugging* necesario para la validacion y corrección de errores.

4.3.2 Evaluador

El evaluador es la función que realiza las *cross-validations*. Para implementar la función evaluator() se ha utilizado KFold[14] para la partición del *dataset*. Como puede verse en el Código 9 el *dataset* se divide y se alterna para calcular el valor de MMRE. Esta partición es aleatoria para cada ejecución del evaluador. Con cada una de las particiones se obtiene el valor de MMRE y finalmente se hace la media de los distintos resultados.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html
[14] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

```python
def evaluator(nfolds, kNN, df, variable):
    kf = KFold(n_splits=nfolds, shuffle=True)
    kf.split(df)
    mmres = []
    for train_index, test_index in kf.split(df):
        df_train, df_test = df.iloc[train_index], df.iloc[test_index]
        mmre = calcular_mmre(variable, df_test, kNN)
        mmres.append(mmre)
    resultado = np.mean(mmres)
    return resultado
```

Código 8 Evaluador

La función del evaluador es llamada por las funciones greedy_forward_selection() y doquire_forward_selection().

4.4 Generación de gráficas [XI]

Para generar las gráficas y los resultados se ha utilizado Jupyter Notebook, donde se importan todos los datos previamente generados y se hacen los cálculos necesarios para generar las gráficas.

La estructura de los resultados generados es la siguiente:

MMRE	k	Variables Elegidas	Metodo	Tiempo	Iteracion

Figura 6 Estructura de datos de los resultados

Valor de MMRE, valor de K, lista de variables elegidas, método, tiempo que ha costado hacer el cálculo y la iteración.

En un primer momento se utilizó matplot, como se puede ver en las gráficas de MI o de mRMR, pero más adelante en el desarrollo, cuando se generó la gráfica de las medias acumuladas de MMRE matplot no era suficiente, por tanto, se planteó buscar alternativas. En una de las reuniones el tutor Fernando González-Ladrón-de-Guevara comentó la posibilidad de utilizar seaborn. Seaborn, como se ha comentado antes, solamente es un complemento de matplot. Provee una API que trabaja sobre matplot y ofrece distintas opciones de personalización de estilo y colores. Haciendo ya por si solo las gráficas mucho más elegantes y legibles.

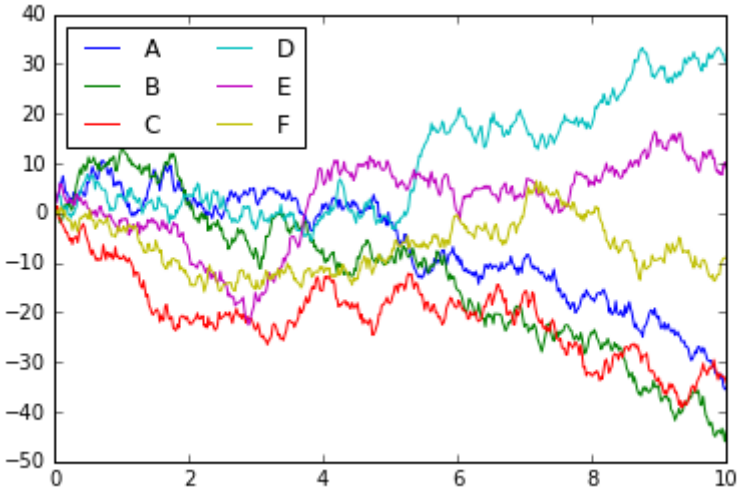

Ilustración 1 Ejemplo gráfica Matplot

Ilustración 2 Ejemplo gráfica seaborn

Para empezar a generar las gráficas lo primero es importar los datos y establecer seaborn como la herramienta de gráficos por defecto. Esto nos permite utilizar las herramientas de generación de gráficas de seaborn de una forma totalmente integrada con pandas, utilizando los métodos de plot de pandas obtenemos gráficos generados mediante seaborn. Lo cual hace que este proceso sea mucho más cómodo ya que simplemente iremos generando *dataframe*s con la información a mostrar y utilizaremos el método plot con los argumentos necesarios para generar la gráfica deseada.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
datak1 = pd.read_csv("500 cross k1.csv", sep=';')
datak2 = pd.read_csv("500 cross k2.csv", sep=';')
datak3 = pd.read_csv("500 cross k3.csv", sep=';')
datak4 = pd.read_csv("500 cross k4.csv", sep=';')
sns.set()
```

Código 9 Inicialización de los datos

Sns.set() nos permite establecer seaborn para que trabaje directamente con los *Dataframe*s de Pandas.

Con la estructura de datos diseñada como resultado de las ejecuciones del algoritmo, se genera 1 columna con todos los valores de MMRE para cada una de las iteraciones. Para acceder a ellas, separamos los datos en los distintos métodos aplicando un filtro, como se ha hecho anteriormente con los proyectos.

```
filtro1 = (datak1['Metodo'] == 1)
filtro2 = (datak1['Metodo'] == 2)
filtro3 = (datak1['Metodo'] == 3)
filtro4 = (datak1['Metodo'] == 4)
data1k1 = datak1.loc[filtro1,:]
data2k1 = datak1.loc[filtro2,:]
data3k1 = datak1.loc[filtro3,:]
data4k1 = datak1.loc[filtro4,:]
```

Código 10 Subconjuntos de datos por método

Una vez tenemos los datos separados por métodos se calcula la media acumulada de la columna de MMRE para cada uno de los métodos. Para esto utilizamos las funciones expanding()[15] y mean() del pandas.*Dataframe*.

```
data_mean1 = data1k1['MMRE'].expanding().mean()
data_mean2 = data2k1['MMRE'].expanding().mean()
data_mean2.index = range(500)
data_mean3 = data3k1['MMRE'].expanding().mean()
data_mean3.index = range(500)
data_mean4 = data4k1['MMRE'].expanding().mean()
data_mean4.index = range(500)
```

Código 11 Calculo de las medias acumuladas.

Para generar la gráfica la forma más sencilla es utilizar un *Dataframe* con los datos.

[15] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.*Dataframe*.expanding.html

```
datafig2_1 = pd.DataFrame({
'MI_1L':data_mean1,
'mRMR_1L':data_mean2,
'MI_2L':data_mean3,
'mRMR_2L':data_mean4})
datafig2_1.plot(figsize=(16, 8))
```

Código 12 Grafica de medias acumuladas

De esta forma generamos la gráfica prácticamente automática, como hemos hecho el sns.set() esta se genera utilizando seaborn, obteniendo una gráfica más agradable visualmente.

El resto de las gráficas y datos generados siguen un proceso más o menos parecido por tanto no voy a comentar como se han generado todas. Pero sí que es interesante el proceso seguido para analizar la elección de variables del algoritmo.

La estructura de datos que sigue la columna de Variables Elegidas no ha sido la mejor, ya que Pandas la identifica como una cadena de caracteres. Por tanto, debemos limpiarla y procesarla antes de poder trabajar con ella. Esto no es un proceso complicado, separamos la cadena por las "," y procedemos a la limpieza de todos los caracteres que no nos interesan.

```
data_variables = data4k1['Variables Elegidas']
final = list()
for item in data_variables.iteritems():
    aux = list(item[1].split(', '))
    linea = list()
    for word in aux:
        word = word.replace('[', '')
        word = word.replace(']', '')
        word = word.replace("'", '')
        linea.append(word)
    final.append(linea)
```

Código 13 Recodificación de las variables elegidas

Con una correcta codificación de los resultados, al realizar la importación en pandas podemos definirlo como que son objetos y por tanto este proceso no sería necesario.

Como se decidió realizar el análisis de proyectos ágiles una vez se tenía totalmente terminado el análisis de los tradicionales, esto se realizó en otro archivo en el que se importaron los datos de ambos y se crearon las gráficas en este.

A la hora de realizar el análisis de los proyectos ágiles se cometió un error al modificar la codificación con el Excel. Revisando los resultados en este se cambió el formato al archivo. Esto provocó que el separador decimal se cambiara de "." a ",", con una búsqueda en Google se obtiene la solución. Pandas nos proporciona un argumento opcional "decimal", en el cual definimos cual será nuestro separador decimal.

```
datak1 = pd.read_csv("Datos Agile.csv", sep=';', decimal=",")
```

Código 14 Importación de datos proyectos ágiles

Una vez realizada esta importación, el proceso para la realización del análisis es sencillo y muy parecido al de los proyectos tradicionales. La única diferencia es que para realizar este análisis se han aprovechado los datos originales para realizar una comparación entre ambos tipos de proyectos.

Simplemente se han generado los *Pandas Dataframes* con los datos necesarios para generar las gráficas. Y si es necesaria algún tipo de configuración para la generación de estas también se ha aplicado, pero en la mayoría de los casos no ha sido así de manera que estas se han creado de una forma muy sencilla.

Para la generación de las gráficas se utiliza un *jupyter notebook* que nos ofrece una gran velocidad de trabajo ya que nos permite ver los valores de las variables en tiempo real y nos permite hacer *plot* de la gráfica en cualquier momento. En caso de error podemos simplemente cambiar como se genera la gráfica y volver a hacer *plot* de esta sin necesidad de ejecutar toda la importación ya que los datos quedan guardados en memoria.

5 Resultados experimentales

5.1 Precisión de los algoritmos de FS

5.1.1 Convergencia de los algoritmos

En primer lugar, la convergencia de los algoritmos es analizada. Para cuantificar la variación, el proceso de *cross-validation* se repite 500 veces. Esto nos permite confiar en la validez estadística de los resultados obtenidos en este trabajo experimental.

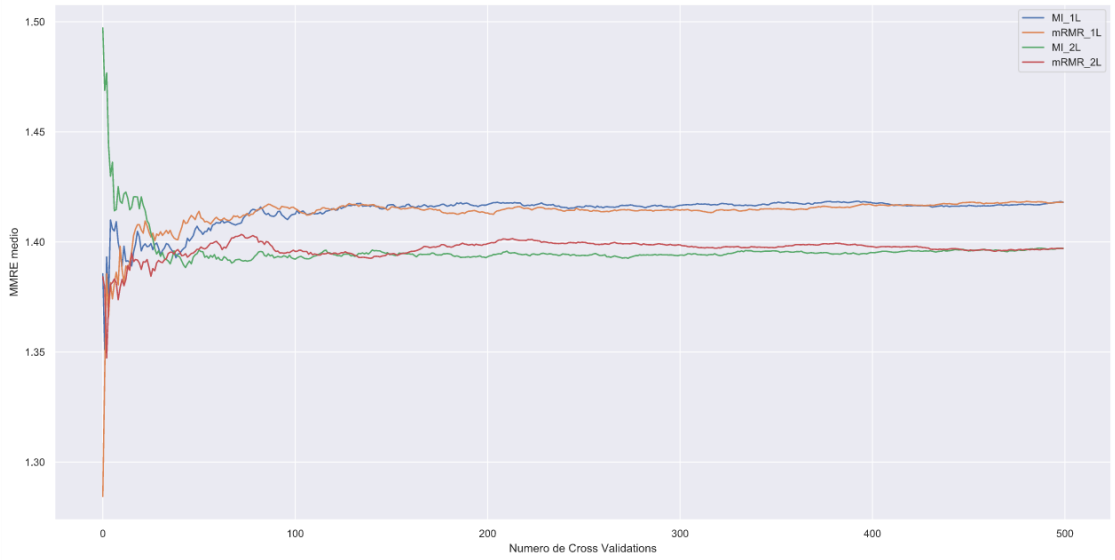

Figura 7 Evolución de las medias acumuladas de MMRE para k=1

La figura 7 muestra la evolución de la media de MMRE a lo largo de las *cross validations* realizadas para cada uno de los algoritmos fijando el valor de k=1. En la figura se puede observar que la media fluctúa mucho a lo largo de las primeras 80 iteraciones, pero tras 250 se estabiliza y la fluctuación disminuye. A partir de unas 350/400 iteraciones se puede decir que el algoritmo se estabiliza prácticamente por completo.

Pero sí que hay una clara diferencia entre los algoritmos que utilizan una lista (MI_1L y mRMR_1L) y dos listas (MI_2L y mRMR_2L). A lo largo de toda la ejecución, con los algoritmos de 2L se obtienen unos valores de MMRE inferiores a los algoritmos que utilizan una única lista. Con estos datos ya podemos empezar a adelantar la conveniencia de los algoritmos que utilizan dos listas.

Para determinar la convergencia del algoritmo las medias acumulativas son comparadas teniendo en cuenta un margen de tolerancia. Se busca que las medias acumulativas de los valores de MMRE cambien menos de un determinado umbral en un número de iteraciones. En la Tabla 4 se puede comprobar el número de iteraciones necesarias para cumplir la condición de convergencia con distintos márgenes de tolerancia cuyos valores oscilan entre el 0.1% y el 0.01%.

Tabla 3 Convergencia de los algoritmos para distintos valores de tolerancia (k=1)

Tolerancia	MI_1L	mRMR_1L	MI_2L	mRMR_2L
0.1%	145	140	138	146
0.09%	158	144	142	158
0.08%	189	177	195	170
0.07%	212	186	208	193
0.06%	271	245	283	230
0.05%	356	298	325	285
0.04%	356	310	366	350
0.03%	412	420	422	414
0.02%	0	0	0	0
0.01%	0	0	0	0

5.1.2 Influencia del valor de K

El mejor valor de k para los vecinos cercanos depende de los datos y de la aplicación de estos. Normalmente este valor se determina experimentalmente. Se ha analizado el rendimiento de los cuatro algoritmos de FS para los distintos valores de k. En nuestro caso, nos interesan los valores más parecidos al objetivo, es decir, valores de k pequeños. Por ello se han seleccionado valores de k de 1 a 4 (1<=k<=4). Realmente si tuviéramos una base de datos muchísimo mayor a la que disponemos podríamos incluso incrementar el valor de K y mejorar la calidad del algoritmo. Con tan solo unos 600 proyectos de muestra y tan heterogéneos si incrementamos el valor de K lo que pasa es que el algoritmo elige proyectos que son muy diferentes unos de otros.

En la tabla 4 se muestra el valor de MMRE para los distintos valores de k teniendo en cuenta los cuatro algoritmos sobre las 500 iteraciones de validación. Los mejores resultados son obtenidos para k = 1. Por tanto, el valor de k de aquí en adelante quedará fijado en k = 1.

Tabla 4 Valores de la media MMRE para los distintos valores de k

K	MI_1L	mRMR_1L	MI_2L	mRMR_2L
1	1.4176	1.41809	1.39706	1.39702
2	1.4895	1.49497	1.4709	1.47229
3	1.5944	1.58901	1.5759	1.5734
4	1.6834	1.6791	1.6671	1.6596

5.1.3 Precisión de los algoritmos de FS

Fijando el valor de k a 1, k=1 la figura 3 nos muestra el *box plot* con la distribución de los valores de MMRE para los cuatro algoritmos. En este podemos observar distribuciones relativamente simétricas. Sin embargo, el valor medio de los algoritmos con 1 lista es más elevado que el de los que utilizan 2 listas.

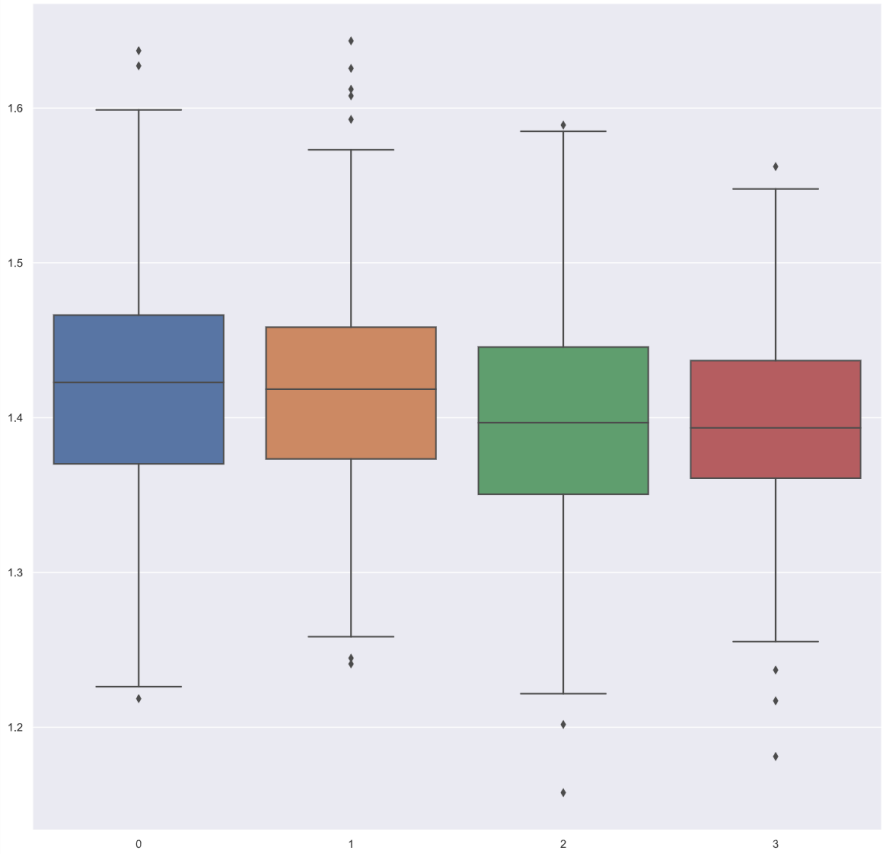

Figura 8 Box plot de la precisión (MMRE) dependiendo de los algoritmos FS

A parte de la precisión de la predicción, el coste computacional también se tiene en cuenta. Los algoritmos están probados en un AMD Ryzen 7 3700X @4.10 GHz y 16Gb de RAM. La tabla 5 muestra la media y la desviación típica de los tiempos de ejecución en 500 iteraciones para cada algoritmo para k=1.

Los *wrappers* son comúnmente criticados por requerir unos niveles de computación muy elevados. En la tabla 5 se pueden observar que los algoritmos que utilizan una lista (1L) tienen unos tiempos de ejecución inferiores a los que utilizan dos listas (2L). También se puede observar que no hay una diferencia substancial entre la utilización de MI o de mRMR. Los tiempos son prácticamente los mismos en este caso, la diferencia está como se ha comentado anteriormente en si se utiliza una única lista o varias.

Tabla 5 Tiempos de ejecución de los algoritmos

Algoritmo	Media (segundos)	Desviación Típica
MI_1L	37.094	0.140
mRMR_1L	37.450	0.309
MI_2L	47.588	4.017
mRMR_2L	47.516	3.958

5.2 Análisis de las variables seleccionadas

5.2.1 Información mutua y redundancia de las variables

Para realizar el cálculo de Mutual Information de las variables del *dataframe*, se separan los valores de la variable objetivo del resto. Una vez realizada esta separación recorremos cada una de las variables dependientes con la variable objetivo utilizando la función de MI.

```
y = df[variable].values
x = df.loc[:, df.columns != variable]
#x = df
resultado = []
for (columnName, columnData) in x.iteritems():
    #print('Colunm Name : ', columnName)
    aux = calc_mi_scikit(y, columnData)
    resultado.append(aux)
resultado = pd.Series(resultado)
resultado.index = x.columns
resultado = resultado.sort_values(ascending=False)
return resultado
```

Código 15 Calculo de MI

Como se puede ver en el Código 16 la función devuelve un pandas.Series[16] con los valores del MI ordenados de mayor a menor. De esta forma posteriormente accediendo al índice de la serie obtenemos una lista con las variables ordenadas.

Para la creación de esta función se han realizado múltiples interpretaciones, quedándonos finalmente con la que más nos ha gustado. En el anexo 8.3 se pueden observar distintas interpretaciones descartadas. El resultado se ha obtenido con la función normalized_mutual_info_score aplicada a todas las columnas del *dataset*.

[16] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

La Figura 9 muestra los valores de MI de las diferentes variables teniendo en cuenta el *dataset* completo. Estas variables son ordenadas de forma descendiente. La lista ordenada nos servirá como entrada de los algoritmos MI_1L y MI_2L.

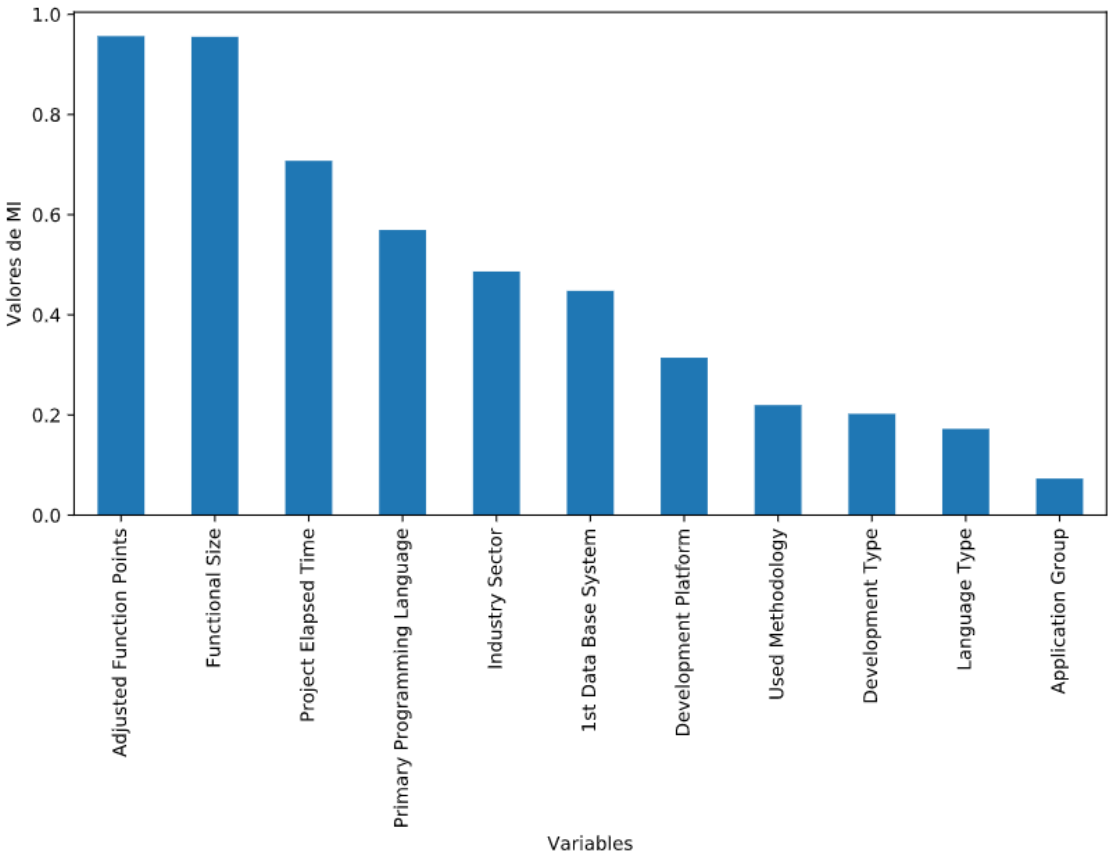

Figura 9 Mutual Information de las variables independientes

El resto de las gráficas se pueden observar en el anexo A del documento.

Como se puede observar en la Figura 9 las variables Adjusted Function Points y Funcional Size tienen el MI más alto respecto a la variable de esfuerzo, seguido de Project Elapsed Time y de Primary Programming Language.

Llegados a este punto también es necesario analizar el orden de las variables teniendo en el segundo criterio, que contempla, además, la redundancia entre las variables dependientes.

Por supuesto como puede observarse en la Figura 9 la primera variable seleccionada es la misma para el criterio de MI. Como AFP tiene un valor de MI muy elevado quiere decir que realmente es una variable muy parecida a la de esfuerzo. Esto hace que al calcular MI para AFP obtenemos unos valores muy similares a los de NWEL1.

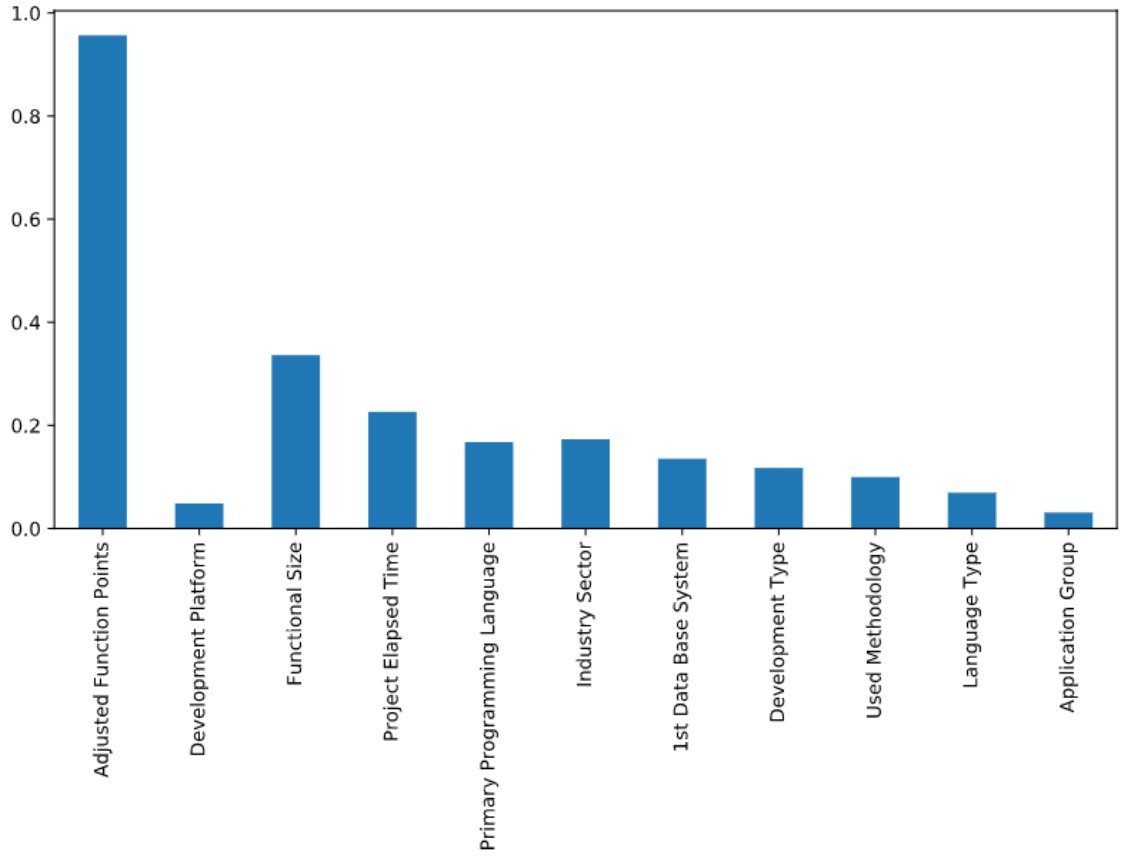

Figura 10 mRMR de las variables independientes

5.2.2 Número de variables seleccionadas

En esta sección se analiza el número de variables seleccionadas por cada algoritmo. Se han hecho 500 iteraciones del algoritmo, en cada una de estas iteraciones se genera un modelo con unas variables determinadas.

Aparentemente los métodos que utilizan 2 listas emplean menos variables para sus modelos que los que emplean 1L, tal y como se puede ver en la Tabla 7. Pero la diferencia no parece ser del todo destacable, aunque si es existente.

Tabla 6 Número de variables seleccionadas por algoritmo

Algoritmo	Media	Desviación Típica
MI_1L	3.84	1.2574
mRMR_1L	3.91	1.2605
MI_2L	4.068	1.36652
mRMR_2L	4.042	1.27455

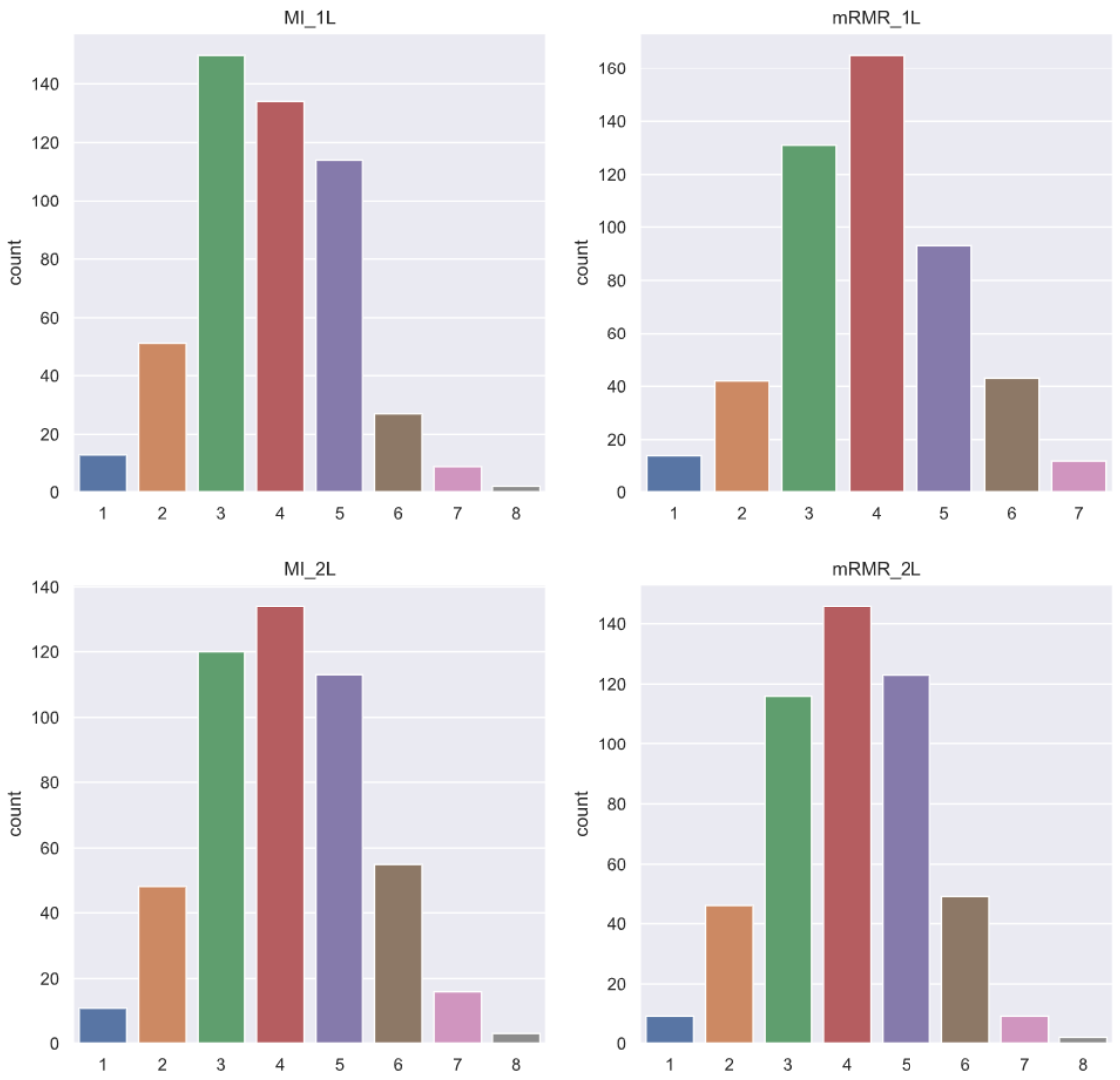

Figura 12 Numero de variables seleccionadas por algoritmo

Para los algoritmos de FS, la Figura 12 muestra la distribución del número de variables seleccionadas, para disponer de más información que la proporcionada en la Tabla 7.

5.2.3 Preferencia de uso de las variables

La preferencia de uso de las variables es analizada en detalle considerando los algoritmos mRMR_1L y mRMR_2L. Las Tablas 8 y 9 nos muestran los resultados del análisis.

La tabla 8 nos muestra la matriz con las frecuencias de uso de las variables independientes, junto con la posición en las cuales estas son seleccionadas por el algoritmo mRMR_1L tras 500 iteraciones k=1. En la primera columna se muestra el número de veces que una variable ha sido elegida en primera posición. En la segunda columna se muestra el número de veces que una variable ha sido elegida en segunda posición, y así sucesivamente hasta la columna 11. En la columna 12 se muestra el número de veces que una variable queda sin seleccionar.

Tabla 7 Uso de las variables seleccionadas por el algoritmo mRMR_1L

mRMR_1L	1	2	3	4	5	6	7	8	9	10	11	12	13	IP
AFP	500	0	0	0	0	0	0	0	0	0	0	0	0	1
PET	0	260	170	0	0	0	0	0	0	0	0	0	70	3,88
FSZ	0	222	0	0	0	0	0	0	0	0	0	0	278	8,116
PPL	0	2	78	58	0	0	0	0	0	0	0	0	362	10,352
IS	0	0	73	51	15	0	0	0	0	0	0	0	361	10,39
1DBS	0	1	40	49	17	2	0	0	0	0	0	0	391	10,996
DP	0	1	35	39	24	4	0	0	0	0	0	0	397	11,136
UM	0	0	7	31	34	8	1	0	0	0	0	0	419	11,634
DT	0	1	15	25	21	7	2	0	0	0	0	0	429	11,774
LT	0	0	12	13	19	10	0	0	0	0	0	0	446	12,082
AG	0	0	7	15	14	3	5	1	0	0	0	0	455	12,254

La última columna de las tablas muestra el valor del índice de preferencia, calculado de la siguiente forma:

$$WPi = \sum_{j=1}^{12} \left(Mi, j^{*} j\right) / \text{runs}$$

Mi,j es el número de veces que una variable i ha sido seleccionada en una posición j

J es la posición en la cual una variable i puede ser seleccionada

Runs número total de iteraciones de *cross validations* (runs=500)

La Tabla 8 nos muestra como la variable AFP es seleccionada en todos los casos en primera posición (IP = 1). La segunda variable preferida es PET que ha sido utilizada 260 veces en primera posición y 170 en segunda. Solo en 70 de las 500 iteraciones esta variable no se selecciona. Por ello su IP es mucho más elevado 3,88. FSZ ha sido utilizada 222 veces en segunda posición. El resto de las variables han sido utilizadas menos veces y sus posiciones varían de forma que no es sencillo sacar una conclusión clara sobre cuales son más o menos apropiadas.

Tabla 8 Uso de las variables seleccionadas por el algoritmo mRMR_2L

mRMR_2L	1	2	3	4	5	6	7	8	9	10	11	12	13	IP
AFP	500	0	0	0	0	0	0	0	0	0	0	0	0	1
PET	0	98	218	57	17	0	0	0	0	0	0	0	110	5,186
DT	0	181	46	34	0	0	0	0	0	0	0	0	239	7,486
FSZ	0	183	35	9	0	0	0	0	0	0	0	0	273	8,112
DP	0	21	66	75	12	0	0	0	0	0	0	0	326	9,676
PPL	0	6	30	40	17	2	0	0	0	0	0	0	405	11,248
IS	0	1	22	32	29	4	1	0	0	0	0	0	411	11,33
UM	0	3	8	24	34	6	0	0	0	0	0	0	425	11,714
1DBS	0	0	5	16	21	14	1	1	0	0	0	0	442	11,816
AG	0	0	5	28	22	9	2	0	0	0	0	0	434	11,894
LT	0	0	7	11	26	10	2	0	0	0	0	0	444	12,082

En la Tabla 9 se muestran los datos relacionados con el algoritmo mRMR_2L. El formato de la tabla es idéntico al anterior. Se puede observar que la variable más elegida vuelve a ser AFP.

A pesar de la utilización de 2 listas una con variables categóricas y otra con variables continuas, las 2 variables más relevantes vuelven a ser AFP y PET. Cosa que por otro lado tiene todo el sentido del mundo, ya que como se ha comprobado a la hora de hacer el algoritmo de *Mutual Information* estas variables contienen mucha más información que cualquier variable categórica. Y por tanto son las que más seleccionan los algoritmos.

Por tanto, la diferenciación en los algoritmos de 2 listas o de 1 lista sí que nos deja una diferencia en el número de variables seleccionadas, así como en las variables que estos seleccionan. Pero realmente no se puede establecer hasta qué punto son representativas estas diferencias. Si es cierto que el MMRE obtiene una mejoría clara con los algoritmos de 2 listas frente a los de lista única. Sin embargo, el coste de computación es mayor, el cual se incrementará conforme nuestra base de datos con experiencias previas de proyectos siga creciendo, requiriendo una mayor potencia de cálculo.

5.2.4 Comparativa entre proyectos tradicionales y proyectos ágiles

Una vez se han obtenido los resultados que corresponden a proyectos desarrollados con metodologías tradicionales, se va a realizar el análisis con una nueva versión de la base de datos, ISBSG – Release May 2017 R1. Con esta nueva base de datos se pretende replicar el análisis anteriormente descrito, pero con los proyectos desarrollados con metodologías ágiles.

El desarrollo ágil de software[17] envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos en ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y las soluciones evolucionan con el tiempo según la necesidad del proyecto. De tal forma que el trabajo es realizado por distintos equipos auto organizados y multidisciplinarios.

En un primer momento se planteó el trabajar exclusivamente con los proyectos desarrollados con metodología ágil. Pero esto se descartó por el número reducido de estos, que condicionaría la fiabilidad de los resultados.

Al igual que en el análisis realizado con los proyectos tradicionales, en esta nueva versión de la base de datos también es necesario una fase de preparación de los datos. El procedimiento es parecido al realizado anteriormente.

Por tanto, en el pre-procesado de datos como es lógico necesitamos quedarnos con los proyectos ágiles. Esto descarta todos los proyectos tradicionales, los cuales no nos interesan para esta segunda parte:

```
filtro = df['Agile Method Used'] == 'Yes'
```

[17] https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software#:~:text=El%20desarrollo%20%C3%A1gil%20de%20software,seg%C3%BAn%20la%20necesidad%20del%20proyecto.

El siguiente filtro selecciona los proyectos cuya variable *Count Approach* es “NESMA”. El cual nos permite establecer que aproximación se ha utilizado a la hora de medir el tamaño del proyecto.

```
filtro = df['Count Approach'] == 'NESMA'
```

Por último, se ha comprobado que el valor de la variable *Sprints / iterations size* no contenga ningún valor NaN. Esto al igual que con los proyectos tradicionales se ha obtenido utilizando la función dropna.

```
df = df.dropna(subset=['Sprints / iterations size'])
```

Al igual que con los proyectos originales no vamos a utilizar todas las variables disponibles en nuestro *dataframe*, ya que la mayoría no contienen información relevante para nuestro objetivo de estimación de esfuerzo. La lista inicial de variables es la siguiente:

'Data Quality Rating', 'UFP rating', 'Industry Sector', 'Application Group', 'Development Type', 'Development Platform', 'Language Type', 'Primary Programming Language', 'Count Approach', 'Functional Size', 'Adjusted Function Points', 'Normalised Work Effort Level 1', 'Summary Work Effort', 'Effort Build', 'Effort Test', 'Effort Implement', 'Project Elapsed Time', 'Business Area Type', '1st Data Base System', 'Used Methodology', 'Resource Level', 'Max Team Size', 'Average Team Size', 'Input count', 'Output count', 'Enquiry count', 'File count', 'Interface count', 'Agile Method Used', 'Sprints / iterations size'

Muchas se repiten con las seleccionadas en el script original, pero otras se incluyen porque pueden ser relevantes para los proyectos ágiles. Al igual que anteriormente comprobaremos el número de NaN que contienen las variables y si no superan el umbral las desecharemos. En este caso todas las variables que contengan un porcentaje mayor al 50% de valores a nulo serán eliminadas.

```
df = df.dropna(axis=1, thresh=int(0.5*len(df)))
```

Posteriormente a esta eliminación de variables, eliminamos los proyectos que contengan valores NaN

```
df = df.dropna()
```

Tras estas eliminaciones nos quedan solamente 49 proyectos.

Con estos proyectos ágiles vamos a seguir el mismo procedimiento descrito sobre los proyectos tradicionales. Pero en este caso se fijará un valor de k = 1, sin realizar el análisis de los valores.

	Data Quality Rating	UFP rating	Industry Sector	Application Group	Development Type	Language Type	Primary Programming Language	Count Approach	Functional Size	Normalised Work Effort Level 1	Summary Work Effort	Effort Build	Effort Test	Effort Implement
210	B	B	Government	Onshore	Enhancement	4GL	Oracle	NESMA	54.0	608.0	428.0	214.0	171.0	43.0
861	A	B	Government	Onshore	Enhancement	4GL	Oracle	NESMA	75.0	641.0	450.0	225.0	180.0	45.0
1020	A	B	Government	Onshore	Enhancement	4GL	Oracle	NESMA	159.0	596.0	418.0	209.0	167.0	42.0
1093	B	B	Government	Onshore	Enhancement	4GL	Oracle	NESMA	88.0	515.0	362.0	181.0	145.0	36.0
1238	A	B	Government	Onshore	New development	4GL	Oracle	NESMA	66.0	241.0	164.0	82.0	66.0	16.0

Figura 13 Vista previa de la BD

Como se muestra en la Figura 13 se han eliminado muchas de las columnas del proyecto ya que no superan los umbrales establecidos y por tanto la lista final de variables es la siguiente. Industry Sector, Application Group, Development Type, Language Type, Primary Programming Language, Functional Size, NWEL1 evidentemente, Summary Work Effort, Effort Build, Effort Test, Effort Implement. Por tanto, nos deshacemos de las variables que nos indican la calidad de los datos de nuestra base de datos, como son Data Quality Rating o UFP rating.

Lo primero la Figura 14 muestra la ordenación de las variables según el criterio de MI.

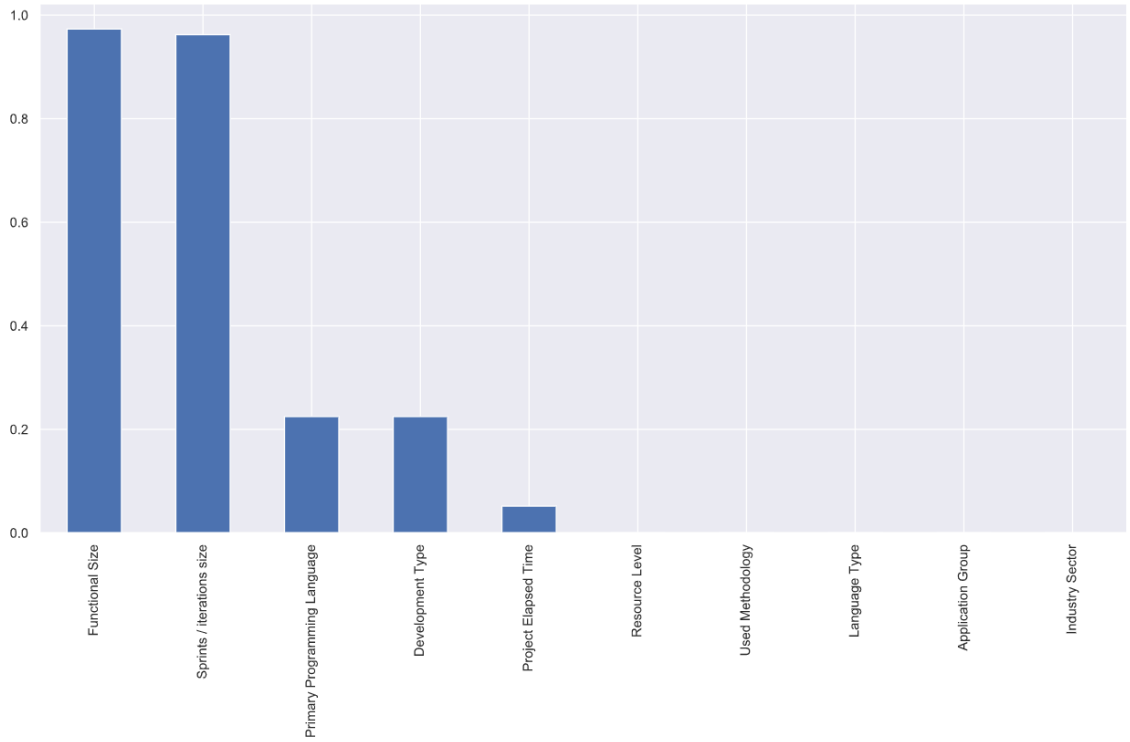

Figura 14 Mutual Information de las variables independientes para proyectos ágiles

Como puede observarse en la Figura 14 no todas las variables contienen información respecto a la variable objetivo. En ningún caso el resultado es 0 pero si es verdad que es un número muy muy pequeño. A partir de aquí, podríamos realmente descartar estas variables, pero por coherencia las vamos a dejar para facilitar las comparaciones.

Al igual que anteriormente la figura 15 muestra la ordenación de variables según el criterio de mRMR.

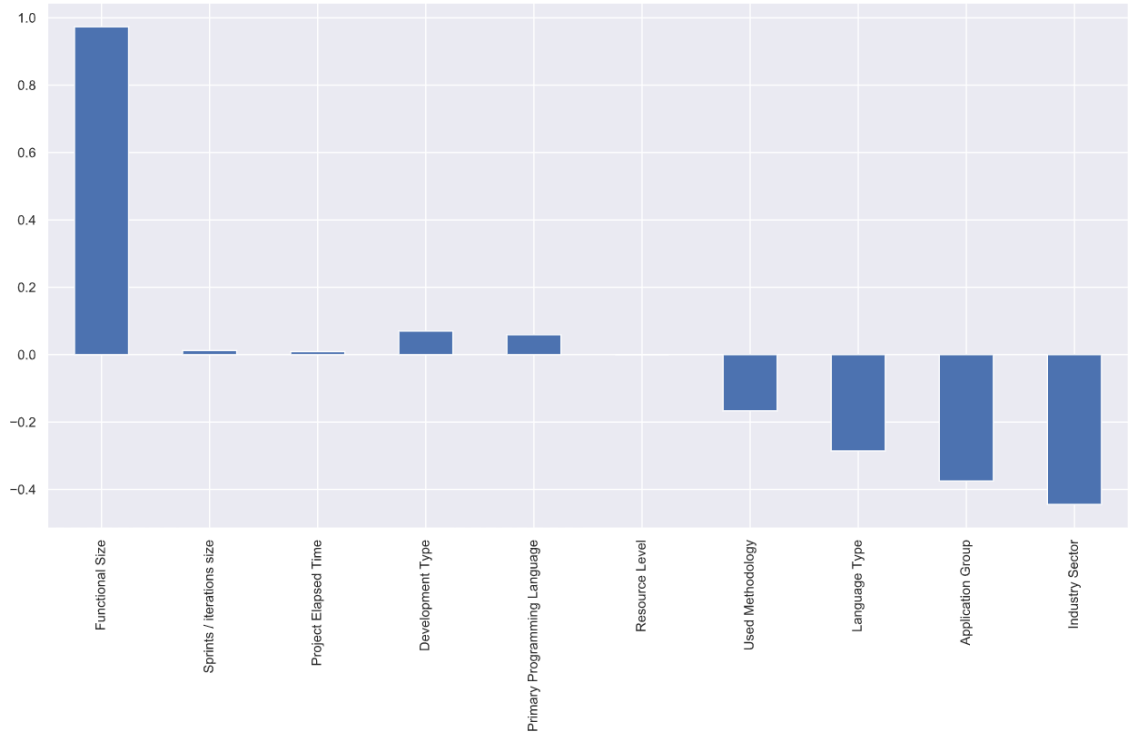

Figura 15 mRMR de las variables independientes para proyectos ágiles

Al igual que con MI no hay ninguna sorpresa respecto a la ordenación de las variables. El orden de estas es coherente y parecido a la experiencia previa con los proyectos tradicionales.

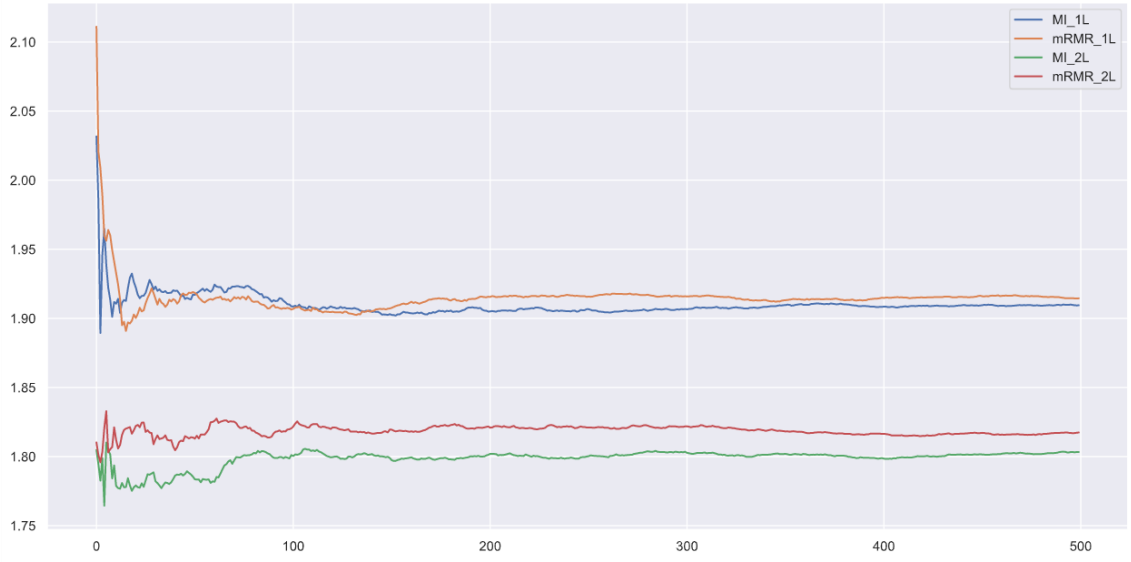

Figura 16 MMRE proyectos ágiles

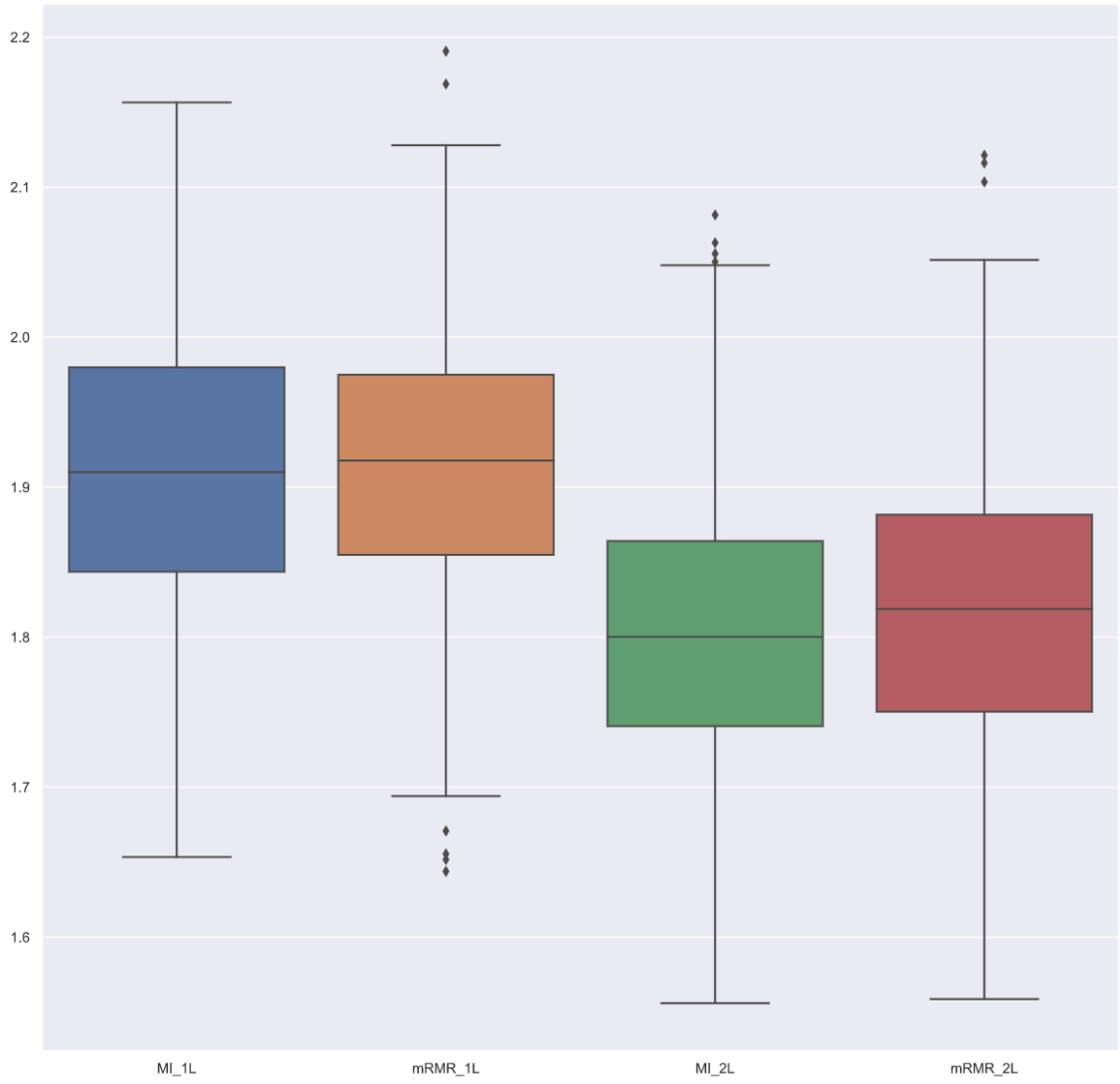

Figura 17 Media MMRE acumulada

En la figura 8 podemos ver el grafico de evolución de los valores de las medias de MMRE acumuladas a lo largo de las 500 iteraciones para k = 1. Como se puede ver los valores oscilan mucho al inicio, lo cual viene dado por el bajo número de proyectos del que disponemos. Dada la naturaleza del algoritmo knn tener un número de proyectos bajo nos genera un mayor error. Las medias acumuladas fluctúan considerablemente a lo largo de las primeras 100 iteraciones. A pesar de que este error es mayor al igual que en el análisis de los proyectos tradicionales parece que la media acumulada se estabiliza entorno a las 200 iteraciones.

Lo que sí se puede ver claramente es la tendencia de que los algoritmos que utilizan 2 listas son mejores que los que solamente utilizan 1 sola lista. Obtienen mejores resultados en el cálculo de mmre lo cual, como ya hemos visto anteriormente nos permite identificar que algoritmo es más eficaz que otro.

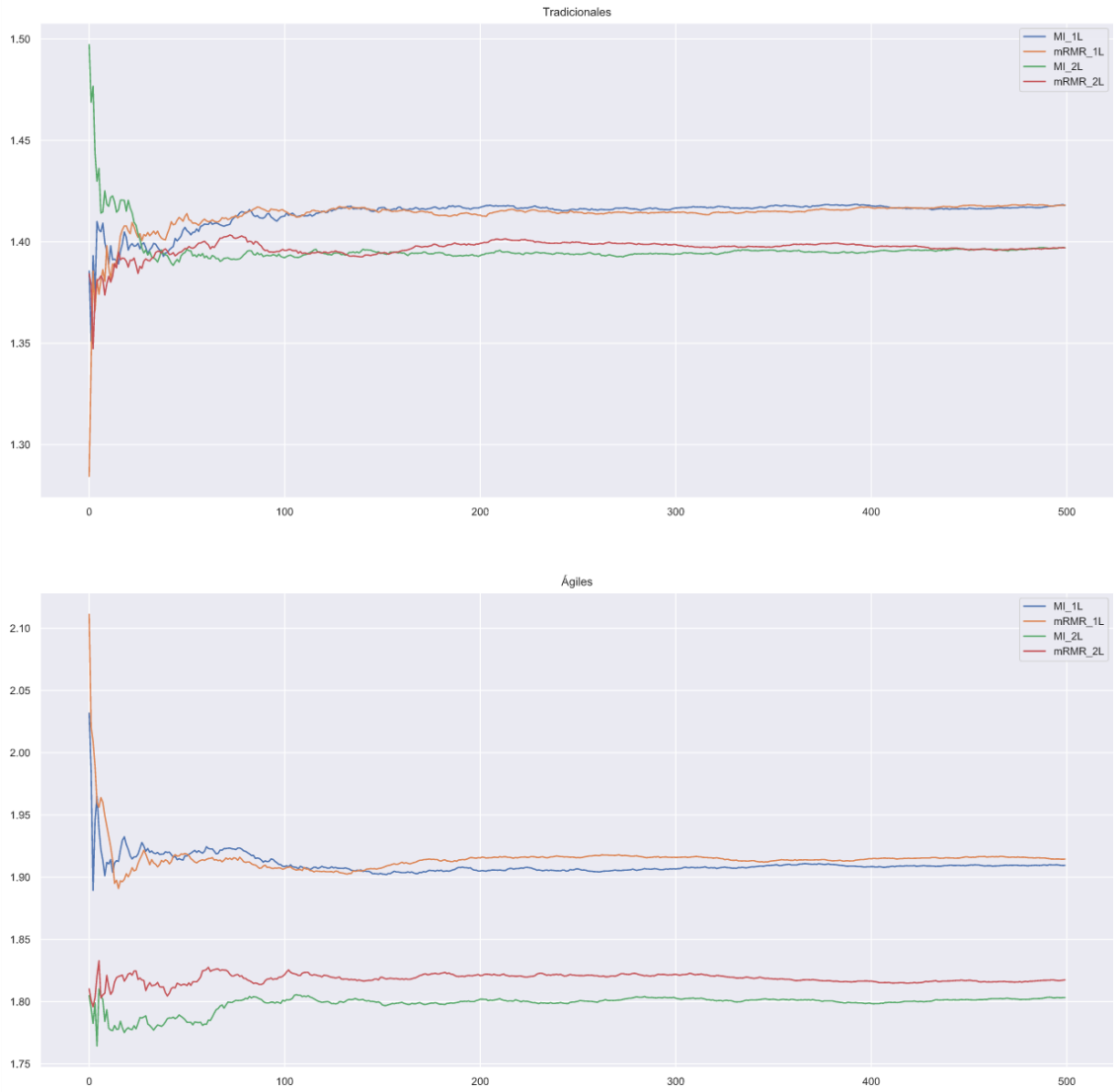

Figura 18 MMRE media acumulativa ágiles y tradicionales

Las figuras 11, 12, 13 y 14 nos muestran los resultados de los distintos algoritmos, para los proyectos ágiles y para los tradicionales.

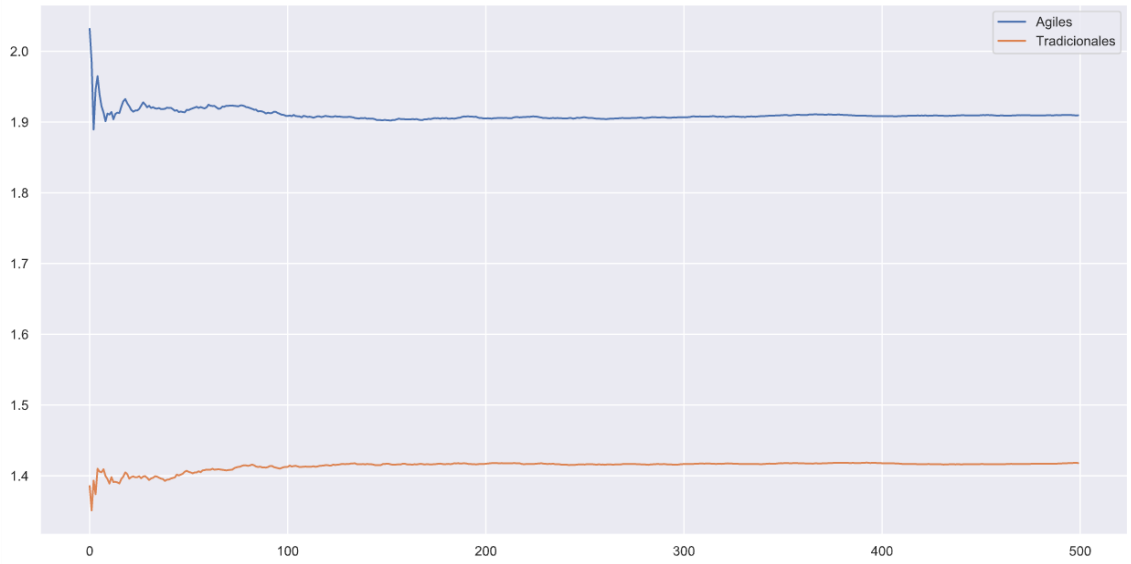

Figura 19 MI_1L

Como se puede ver en la Figura 10, donde podemos observar las medias acumuladas de MMRE del algoritmo MI_1L con los proyectos ágiles y los desarrollados utilizando un método tradicional, en los proyectos ágiles no solo tenemos un mmre más alto, sino que también tarda más iteraciones en estabilizarse. Esto se debe a la menor cantidad de proyectos de los que disponemos. Esta menor cantidad de proyectos dificultan la tarea al algoritmo.

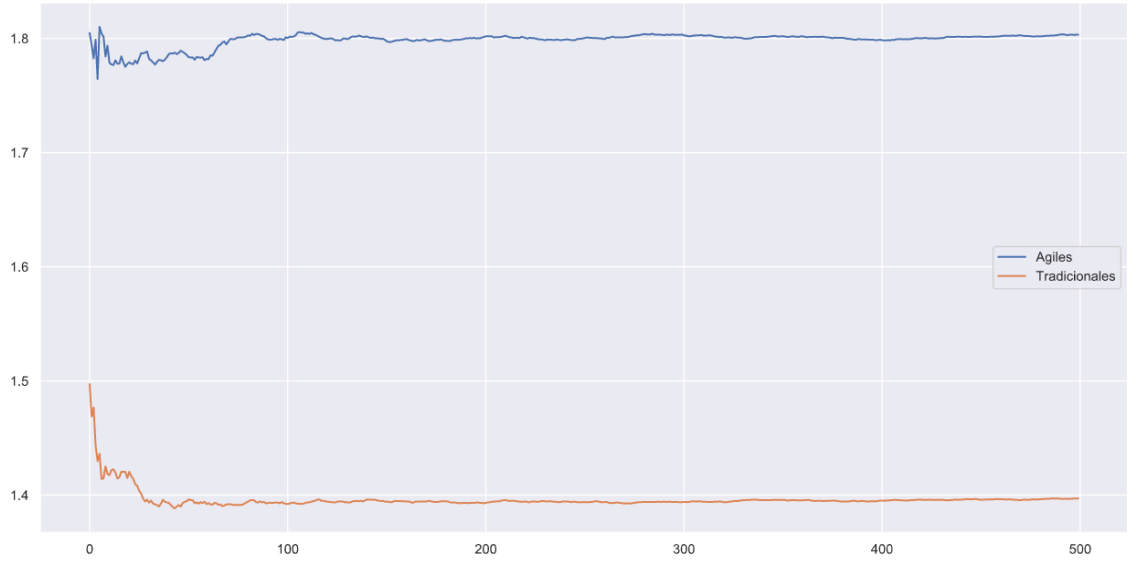

Figura 20 MI_2L

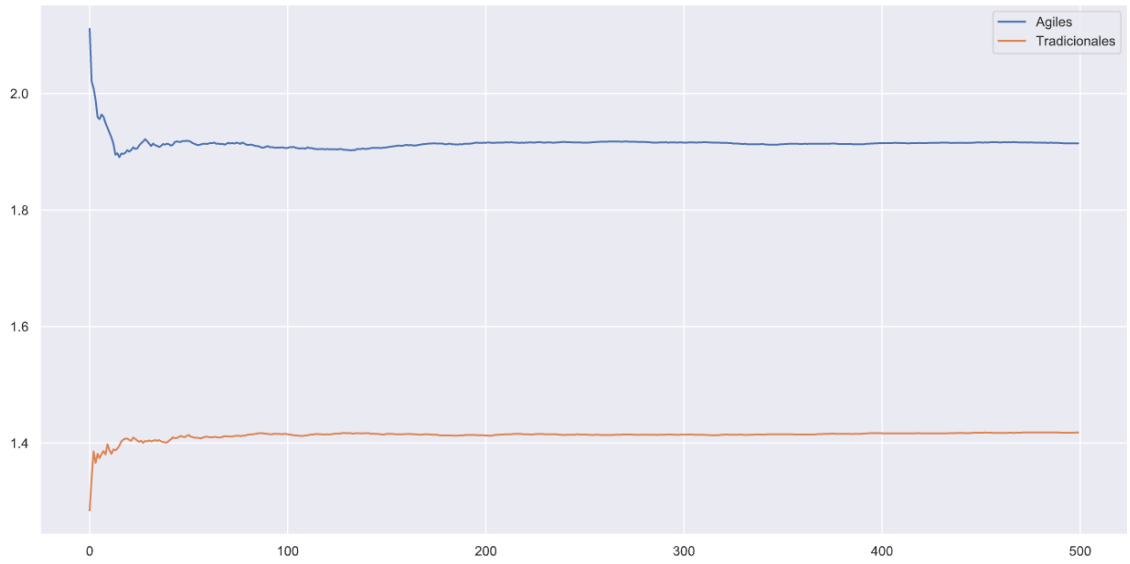

Figura 21 mRMR_1L

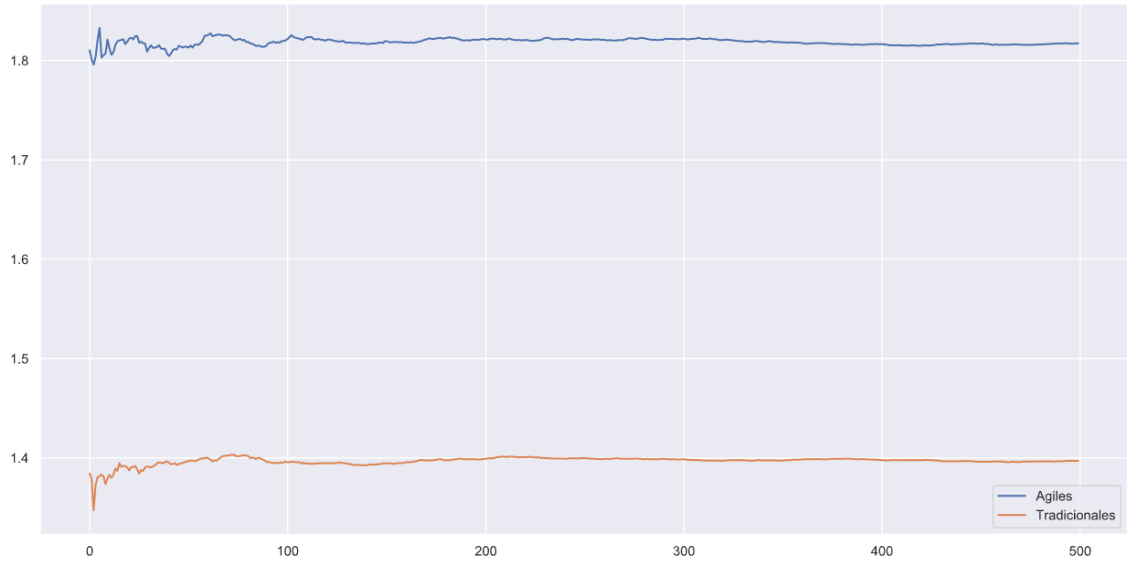

Figura 22 mRMR_2L

Como se puede ver en las figuras los resultados son prácticamente idénticos y concuerdan con los resultados esperados.

Al igual que en el trabajo de análisis anterior, en esta nueva ejecución sobre los proyectos ágiles también se han recogido los tiempos de ejecución. Pero como es lógico tampoco se espera una diferencia notable en los tiempos ya que dada la naturaleza del algoritmo no tiene una gran importancia el tipo de proyectos que se eligen. Aun así, hay una pequeña diferencia en los tiempos que debe venir dada por el menor número de proyectos del que se disponen.

Tabla 9 Tiempos proyectos ágiles y tradicionales

Algoritmo	Ágiles	Tradicionales
MI_1L	35.491	37.094
mRMR_1L	36.322	37.450
MI_2L	44.026	47.588
mRMR_2L	43.928	47.516

Bien es cierto que el resultado de la selección de variables no ha sido todo lo preciso que se requiere. Esto una vez más viene dado por la escasez de proyectos, con tan solo unos 50 proyectos es complicado hacer funcionar un algoritmo de knn. No en el número de las variables, pero si en la selección y el orden de estas. Como se puede ver en la Figura 8 algunas de estas variables no contienen nada de información con respecto a la variable objetivo. Por tanto, la inclusión de estas a la hora de ser seleccionadas por el algoritmo solamente se debe a una casualidad. Es difícil asegurar que la inclusión de estas variables realmente aporta una mejoría clara de forma consistente.

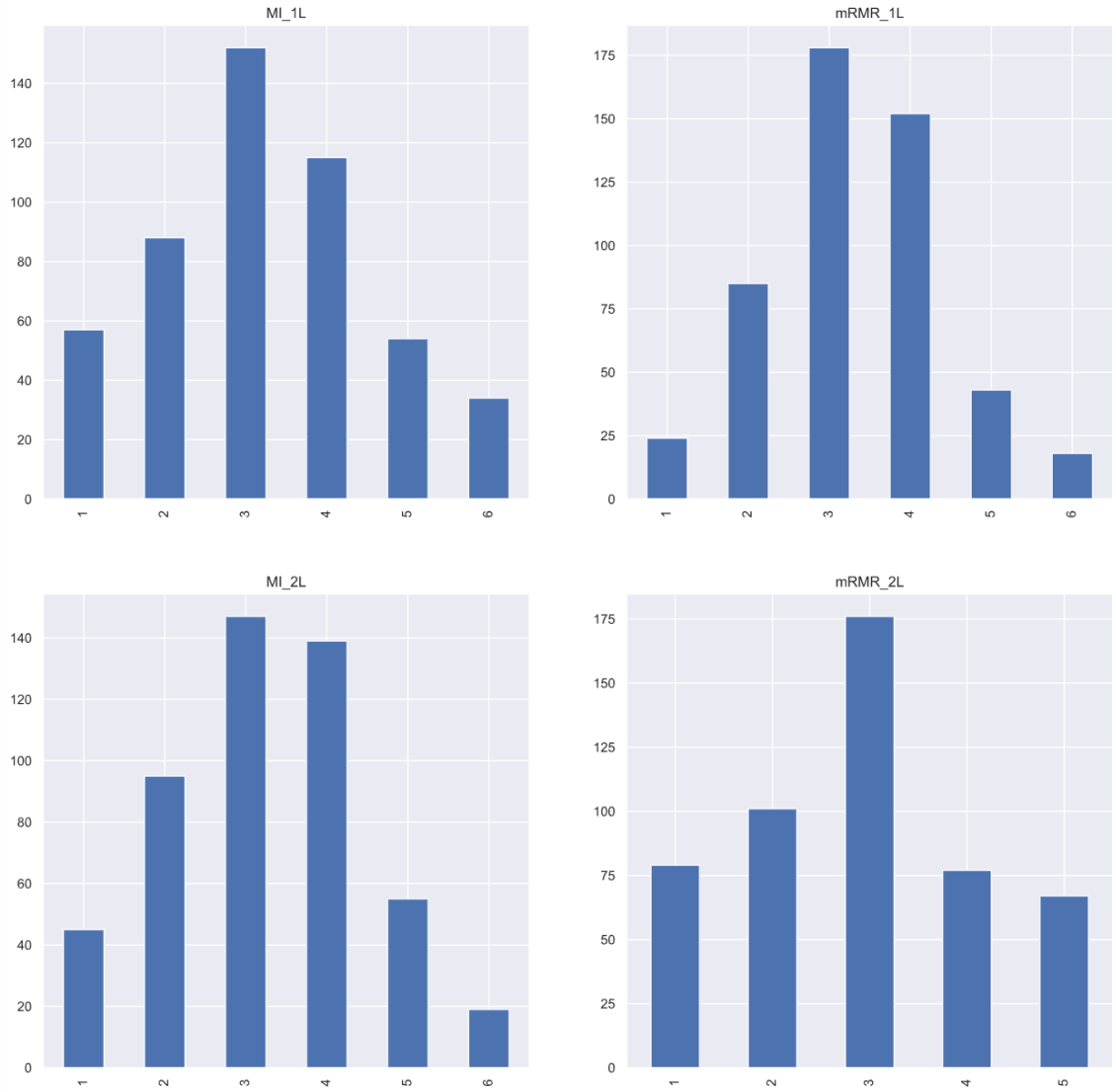

Figura 23 Número de variables seleccionadas por algoritmo

Como se puede ver en la figura 17 la cantidad de variables seleccionadas sigue una distribución normal[18].

Si que es verdad que las variables más seleccionadas son aquellas que contienen más información según el algoritmo de *mutual information*. Pero también hay una cantidad muy elevada de variables que son elegidas aparentemente de forma totalmente aleatoria siendo imposible obtener alguna conclusión certera sobre la utilidad de estas. Aunque es mera especulación parece haber una tendencia respecto a que las variables categóricas contienen menos información que las numéricas, y por tanto son elegidas una menor cantidad de veces. Pero sí que parece haber una clara tendencia a que las variables con información relacionada con el tamaño del desarrollo son las que más información nos aportan a la hora de generar nuestro modelo. Cosa que por otro lado es totalmente lógica a un mayor tamaño de desarrollo se requiere un mayor esfuerzo.

A falta de una base de datos más grande con una mayor cantidad de proyectos desarrollados utilizando metodología ágil, podemos intuir que estos no son tan diferentes de los tradicionales. Ya que las variables elegidas por nuestro algoritmo son prácticamente iguales. También es cierto que debemos puntualizar una serie de claves para dar un contexto más preciso a esta afirmación. Debemos tener en cuenta que no trabajamos con todas las variables de nuestra base de datos, esta contiene variables que son exclusivas de los proyectos ágiles, pero aún no nos son de utilidad ya que en muchos casos esa información no está disponible. Por tanto, afirmar a ciencia cierta que podemos estimar esfuerzo de proyectos ágiles de la misma forma que lo hacemos de aquellos que utilizan desarrollo tradicional no es viable. El hecho de estar utilizando prácticamente las mismas variables en ambos casos nos hace que a la hora de el algoritmo realizar la selección esta sea muy similar. Utilizando solamente el sentido común y teniendo unos conocimientos mínimos de cómo funciona el desarrollo de software, o en general el desarrollo de cualquier producto, logramos comprender que el tamaño de este está directamente relacionado con el esfuerzo que nos llevará desarrollarlo.

En futuras versiones de la base de datos sería muy interesante realizar un análisis y una ejecución de este algoritmo utilizando las variables relacionadas con el desarrollo ágil. Aunque como se ha comentado anteriormente es necesario disponer de esta información ya que en esta versión en la mayoría de los proyectos esta información se encuentra vacía.

Según mi escasa experiencia en el mundo laboral, el desarrollo guiado por la utilización de metodologías ágiles tampoco es la panacea. Con el desarrollo ágil no se ha descubierto la octava maravilla del mundo. Pero con esto no me refiero a que este sea inútil, todo lo contrario, estas metodologías ayudan a los desarrolladores, a los jefes de proyecto, a los líderes de equipo… Estas metodologías nos permiten reducir los bloqueos que ocurren durante el desarrollo software, nos permiten una mayor adaptabilidad y al fin y al cabo mejorar el rendimiento de nuestros propios equipos. Por tanto, sí que tenemos una diferencia, una mejoría, pero no es una mejora tan radical como para pensar que habrá unas grandes diferencias en las estimaciones que debemos hacer.

[18] https://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal#Historia

6 Conclusiones

6.1 Principales aportaciones

En este trabajo se ha realizado el análisis de la base de datos de ISBSG con el fin de extraer un extra de información que a simple vista no está visible. Este análisis puede resultar de utilidad para futuros proyectos. Este análisis se ha realizado sobre más de 1 versión de la base de datos por tanto con el fin de realizar futuros trabajos con las bases de datos de ISBSG se puede analizar la información de las variables. En este trabajo previo de análisis de la BD se ha analizado más de una versión esta. Dependiendo de las características de esta se ha orientado el análisis en una dirección o en otra. Una de las versiones de la base de datos contiene información sobre proyectos ágiles. Con el fin de poder ser utilizada se realizó el análisis para ver de qué manera podíamos obtener la mayor cantidad de proyectos útiles. Pero esto se descarta ya que en el mejor de los casos obtenemos unos 50 proyectos, que teniendo en cuenta que la BD completa son sobre 8000 es un número bajo. Si bien es cierto no se dispone de la última versión de la base de datos de ISBSG, es muy probable dada la tendencia del desarrollo que en las nuevas versiones se disponga de más proyectos de desarrollo ágil.

A pesar de que la librería Pandas es ampliamente utilizada por internet se ha realizado una amplia utilización de sus herramientas con el fin de realizar un trabajo relacionado con la información mutua.

Se ha creado una librería en Python totalmente documentada y reutilizable relacionada con los algoritmos de *mutual information*. Por tanto, esta librería puede ser utilizada y modificada para la realización de futuros trabajos en Python. Además, en esta misma librería se ha realizado la interoperabilidad de Python con R. Por lo que podríamos decir que la librería select-features.py contiene las funciones necesarias para hacer el análisis tanto en Python como en R y el usuario final puede elegir que lenguaje utilizar. Esto nos permite utilizar R desde Python, lenguaje que tiene una sintaxis mucho más fácil de utilizar.

6.2 Relación con los estudios cursados

Como se puede observar en este trabajo se mezclan conceptos de ciencia de datos, con algunos de ingeniería del software. De hecho, gracias a este trabajo he aprendido una cantidad inmensa de conceptos relacionados con la ciencia de datos y el *machine learning* que no se obtienen durante el grado.

Por desgracia cuando yo empecé en la UPV no estaba disponible el grado en ciencia de datos. Realmente este trabajo hoy en día estaría más relacionado con esta.

Aun así y pese a que en la carrera no se cursan contenidos relacionados con la ciencia de datos sí que hay contenidos relacionados. Durante el desarrollo de todo el código se ha utilizado un sistema de control de versiones de tal forma que se ha podido realizar trabajo en paralelo a la hora de ir haciendo pruebas. También se ha documentado el código siguiendo los estándares.

6.3 Limitaciones del trabajo

El trabajo tiene una serie de limitaciones que merece la pena comentar. El objetivo del trabajo que se planteó durante las primeras semanas era el de comparar los distintos algoritmos con los proyectos desarrollados con tecnología ágil. Pero cuando se realizaron los análisis de las bases de datos llegamos a la conclusión de que no tenían suficientes datos de proyectos de calidad. Dada la naturaleza del algoritmo utilizado es esencial disponer de cuantos más datos de calidad posibles.

Esto es uno de los problemas que tiene la base de datos de ISBSG, la cual contiene una cantidad muy elevada de proyectos que no son válidos. Este es un pensamiento algo egoísta, el hecho de que una base de datos con más de 6000 entradas y con más de 100 variables diferentes no contiene información de calidad. Pero para el desarrollo del proyecto esto es así, una vez filtrados los proyectos según nuestro criterio no quedan más de 620 proyectos, solamente un 10% de la base de datos son proyectos válidos.

Evidentemente se hace el análisis de los 4 algoritmos propuestos en este trabajo, pero se dejan una cantidad muy elevada de algoritmos de *Feature Selection*. Sobre los cuales puede ser interesante realizar la comparativa, tanto con 1 lista como con 2.

Por otro lado, me gustaría comentar una serie de variables que nos dejamos en el tintero que mediante mera especulación creo que podrían ser interesantes. Me gustaría que para el trabajo se dispusiera de variables relacionadas con la experiencia del equipo de desarrollo. Pienso que es un factor tremendamente importante en relación con cómo se trabaja en estos y en la eficiencia de estos. Jose Ignacio Ajenjo, Director de Producción de Schneider Electric España siempre me ha dicho: *"A una maquina si tú le pides que haga A siempre hace A, pero a una persona si le pides hacer A unas veces hará A, unas veces mejor otras peor... Pero otras veces le pedirás hacer A y esta hará B o directamente puede que no haga nada."* Con esto se refiere a que al contrario de las máquinas las personas no siempre tenemos las mismas respuestas ante las mismas peticiones, ya que hay una serie de factores absolutamente incalculables con relación al estado de cada persona. Con esto quiero decir lo siguiente, nos estamos dejando una serie de variables interminables con relación a las personas, a los equipos que realizan el desarrollo de estos proyectos que estamos estimando. Tras una cantidad muy elevada de horas de investigación en la realización de este proyecto creo que en casi ningún sitio se le da una gran importancia a las personas que realizan estos trabajos y por tanto hay una parte extremadamente importante que queda fuera del foco de estos trabajos.

Desde mi punto de vista la unión hace la fuerza y tanto las características de los proyectos como las características de los equipos de desarrollo son cruciales para realizar una estimación de esfuerzo. Podríamos realizar primero un análisis de nuestro equipo de desarrollo por ejemplo realizando en los proyectos iniciales la estimación de esfuerzo utilizando íntegramente los datos con relación al proyecto. Una vez tenemos nuestro *feedback* podemos comparar cual ha sido el esfuerzo requerido para la realización de estos proyectos con el esfuerzo estimado en primera instancia. Podríamos llegar a la conclusión de que quizá nuestro equipo es más productivo de lo esperado, o que por el contrario es menos productivo, o que nos es más sencillo realizar los proyectos que tienen una serie de características en concreto que otros... Pero creo firmemente que si alimentáramos nuestro modelo tanto con información de los proyectos como con

información de los equipos obtendríamos un resultado considerablemente más preciso que utilizando solamente información de los proyectos.

Creo que no hace falta comentar muy en detalle esto, pero de manera evidente con información solamente de los equipos de desarrollo no se puede realizar una estimación de esfuerzo.

6.4 Trabajos futuros

Si bien es cierto que el trabajo es completo y puede darse por terminado, hay una serie de cosas que se pueden realizar. Durante el inicio del trabajo en la fase del planteamiento se tenían varias versiones de la base de datos de ISBSG. La primera intención que se tuvo era la de realizar este trabajo de imputación con los proyectos desarrollados con metodología ágil, de forma exclusiva. Por desgracia esto se descartó ya que tras un exhaustivo análisis no se tenían suficientes proyectos para realizar este trabajo. Esto se podría realizar con las próximas versiones de la BD de ISBSG, la cual cada vez contiene una mayor cantidad de proyectos agiles. Esto es por la tendencia actual donde cada vez es más frecuente el hecho de trabajar y realizar los desarrollos con metodología ágil. El hecho de trabajar con proyectos ágiles nos permitiría realizar un nuevo análisis de las variables seleccionadas lo cual nos permite ver si hay diferencias con los proyectos con desarrollo tradicional. Así como podríamos observar el comportamiento del algoritmo a la hora de realizar las imputaciones. Quizá podríamos observar una precisión más elevada o no, por el contrario, si realizáramos el estudio con proyectos agiles encontraríamos un algoritmo peor. Aunque y esto es mera especulación, no creo que encontráramos grandes diferencias, ya que sí cambia la filosofía de trabajo, pero el trabajo a realizar es el mismo. Si bien es cierto sería más interesante el poder comprara la realización de un mismo proyecto realizado mediante desarrollo tradicional o mediante un desarrollo ágil. Pero esto es prácticamente imposible ya que la experiencia del equipo de desarrollo es también un factor muy importante a la hora de realizar un trabajo.

Cambiando un poco el foco de trabajo, durante la realización de este trabajo se ha experimentado con la ejecución concurrente en Python con el fin de acelerar la realización de los cálculos necesarios. Desgraciadamente la falta de tiempo ha impedido el generar una solución válida utilizando este método. Como se comenta en los posteriores anexos se ha trabajado con el multiprocesado en Python. Aun así, a pesar de que no se ha llegado a una solución completa, el script se vio modificado con el fin de poder ser ejecutado en paralelo junto con otras versiones de este. Por ejemplo, como en el trabajo se realizan cálculos para distintos valores de k, se modificó el script de tal manera que se pudieran lanzar estos cálculos a la vez, acortando el tiempo haciéndolo cuatro veces más rápido.

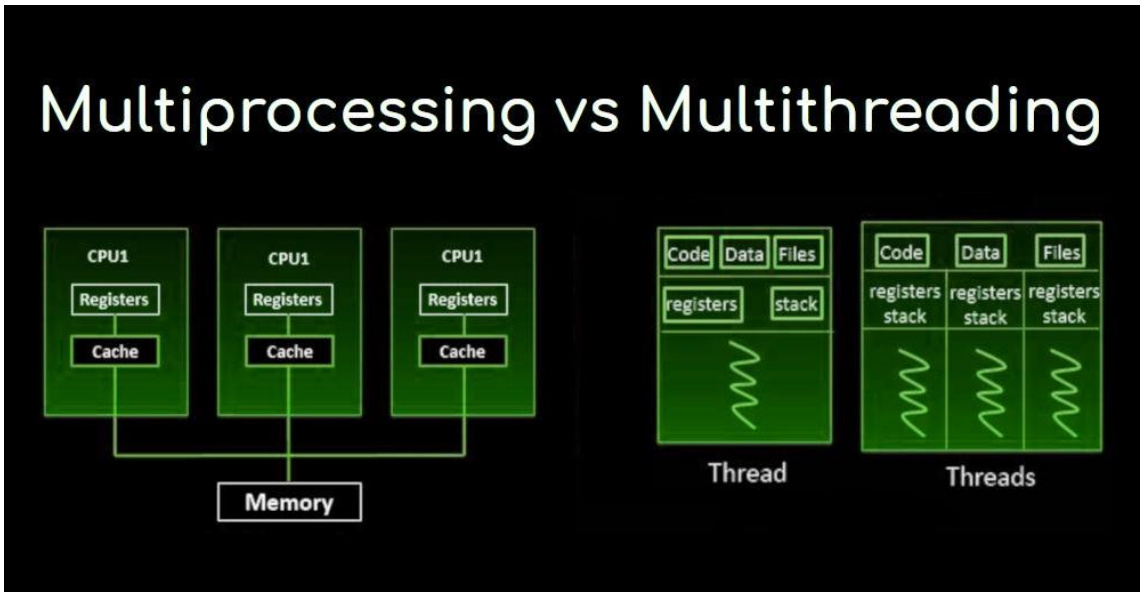

Ilustración 3 Multiprocesado y Multihilo

Además de producir los datos de forma concurrente existen una serie de mejoras posibles a realizar. Para la producción de las gráficas como se ha comentado anteriormente se utiliza un jupyter notebook, pero esto puede cambiar. Se podría generar un documento correctamente formateado y en un formato más legible para un posible usuario final.

Utilizando el paquete Pillow se pueden guardar imágenes, en nuestro caso las gráficas generadas con seaborn, en formato PDF. Por tanto, podríamos generar un informe correctamente formateado con los resultados obtenidos.

También se podría llegar a vender como servicio web. Tras comprobar la documentación de la base de datos no creo que fuera posible utilizar sus datos con el fin de vender un servicio. Pero si obtenemos datos de otras fuentes no sería un problema.

En caso de disponer de una BD con información relacionada con los equipos de desarrollo deberíamos de alguna manera aprender a utilizarla para estimar esfuerzos. Esto debería ser un trabajo conjunto para el cual sería interesante disponer de psicólogos y de expertos líderes de equipo con una amplia experiencia. Creo que sería igual de complicado el hecho de obtener información de como de bien o mal trabaja un equipo de desarrollo que el propio hecho de aprovechar esta información en nuestro objetivo. Pero aun así pienso que sin aprovechar toda la información respecto a los equipos de desarrollo estaríamos perdiendo una gran cantidad de información.

7 Referencias bibliográficas

[I] Web de ISBSG
https://www.isbsg.org/

[II] The usage of ISBSG data fields in software effort estimation Fernando González-Ladrón-de-Guevara, Marta Fernández-Diego & Lokan, C.

[III] A Review of Studies on Expert Estimation of Software Development Effort ISESE. Jørgensen, M

[IV] A review of software surveys on software effort estimation ISESE. Molokken, K. Jorgensen, M.

[V] Página de Wikipedia con información de Feature Selection
https://en.wikipedia.org/wiki/Feature_selection

[VI] Web de scolarpedia con información de Mutual Information
http://www.scholarpedia.org/article/Mutual_information#:~:text=Mutual%20information%20is%20one%20of,variable%20given%20knowledge%20of%20another.

[VII] Application of mutual information-based sequential feature selection to ISBSG mixed data. Marta Fernández-Diego, Fernando González-Ladrón-de-Guevara.

[VIII] Web de Towards data science
https://towardsdatascience.com/

[IX] Web de Medium.com
https://medium.com/

[X] Evaluación del algoritmo kNN para imputación de datos. Alfonso Polo Serrano

[XI] Documentación y tutoriales Seaborn
https://seaborn.pydata.org/tutorial.html

[XII] Libro Python Data Science Handbook
https://jakevdp.github.io/PythonDataScienceHandbook/

[XIII] Documentación oficial Python
https://docs.python.org/3/

8 Anexos

8.1 Introducción a Pandas [XII]

Pese a tener una base de Python decente, nunca había trabajado con nada relacionado con el *DataScience*. Por tanto, con una simple búsqueda en Google con las palabras claves Python y DataScience llegas a Pandas. También vas a encontrar conceptos como Anaconda o Matplotlib, de los cuales ya se ha comentado su uso anteriormente.

Para comprender como funciona Pandas es necesario conocer un poco que es NumPy, y en concreto su objeto ndarray. NumPy es una librería para Python que da soporte a un con junto de objetos y a una serie de funciones matemáticas de alto nivel. Pandas, funciona sobre NumPy aprovechando las funcionalidades que esta añade. Pandas ofrece una implementación eficiente de un *Dataframe*. Los *Dataframe*s son básicamente un array multidimensional con etiquetas tanto de filas como de columnas acoplados. Además de añadir una forma eficiente de etiquetar datos, ofrece una serie de operaciones muy útiles a la hora de trabajar con estos *Dataframe*s. En Pandas no solo tenemos los *Dataframe*s, también tenemos las Series, que pueden verse como *Dataframe*s de 1 sola columna, aunque tienen algunas diferencias con estos.

Para instalar Pandas podemos comprobar la documentación oficial [19] donde encontraremos el método más actualizado para hacerlo. En caso de utilizar Anaconda esto no será necesario ya que pandas viene instalado por defecto. Una vez instalado solo tendremos que importarlo en nuestro programa. Normalmente se importa Pandas bajo el alias pd.

import pandas as pd

En la mayoría de las ocasiones esta se importa junto con NumPy, la cual puede ser importada como:

import numpy as np

8.1.1 Pandas Series Object

Un Pandas Series es un array unidimensional de datos indexados o etiquetados. Esta puede ser creada desde una lista o un array.

data = pd.Series([0.25, 0.5, 0.75, 1.0])

```
0    0.25
1    0.50
2    0.75
3    1.00
```

[19] https://pandas.pydata.org/

Como se puede ver en la salida de nuestro programa, la Series tiene dos secuencias de valores, los cuales pueden ser accedidos mediante los atributos values y index. Los values son simplemente un NumPy array.

data.values

```
array([ 0.25,  0.5 ,  0.75,  1.  ])
```

Mientras que index es un objecto parecido a un array de tipo pd.Index.

data.index

```
RangeIndex(start=0, stop=4, step=1)
```

Al igual que en un array, en Python los datos pueden ser accedidos utilizando el índice.

data[1]

```
0.5
```

data[1:3]

```
1    0.50
2    0.75
dtype: float64
```

Los Pandas Series en general son mucho más flexibles que el NumPy array unidimensional al cual emulan.

La gracia que tienen las Series es este atributo Index el cual puede ser modificado a nuestro antojo. Además, estas pueden ser del tipo que queramos, por ejemplo, se pueden utilizar strings.

data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd'])

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

Y evidentemente la forma de acceder a los datos es como se podría esperar.

data['b']

```
0.5
```

Evidentemente los índices pueden ser no continuos o no secuenciales, como por ejemplo 2, 6, 4, 7.

```
2    0.25
```

```
5    0.50
3    0.75
7    1.00
dtype: float64
```

Otra forma de pensar en un Pandas es como si fuera un diccionario de Python. Un diccionario es una estructura de datos que mapea un cierto valor con unas claves determinadas. En este caso el Pandas Series ofrece una aproximación más eficiente que el diccionario para un cierto tipo de operaciones.

dictionary = {'A':10, 'B':20, 'C':30}

series = pd.Series(dictionary)

```
A      10
B      20
C      30
dtype: int64
```

A diferencia de los diccionarios en las Series se pueden utilizar técnicas de arrays, como el slicing o particionado.

Por tanto hemos visto distintas formas de generar un pd.Series.

pd.Series(data, index=index)

En el caso anterior el argumento index es totalmente opcional y en caso de no indicarse los índices serian números en orden ascendente. Pero como hemos comentado anteriormente los datos de una Series puede ser un diccionario. En cuyo caso los índices son las claves del diccionario ordenadas.

8.1.2 Pandas *Dataframe* Object

Si una Series es similar a un array de 1 dimension, un *Dataframe* es un array de 2 dimensiones con índices de filas y nombres de columnas flexibles. Realmente durante el trabajo el *Dataframe* es la pieza más importante de esta librería.

Al igual que con las series hay múltiples formas de crear un *Dataframe*. Por ejemplo, se pueden crear utilizando series que contienen los mismos índices, asignándose cada una de estas series a una columna determinada.

Al igual que en las Series, un *Dataframe* tiene un atributo index, donde al igual que en las series obtenemos los índices de las distintas filas del *Dataframe*. Además, los *Dataframe*s tienen un objeto columns el cual contiene los nombres de las columnas del *Dataframe*.

Es interesante la opción de crear un *Dataframe* a partir de un archivo csv correctamente formateado, al fin y al cabo, un *Dataframe* no deja de ser un array de dos dimensiones el cual puede ser interpretado como una tabla. Por tanto, con un único método podemos obtener un *Dataframe* totalmente válido sobre el cual empezar a trabajar.

8.1.3 Selección de datos

Tener distintas estructuras de datos para almacenar estos de la forma que más nos convenga está bien, pero si no fuéramos capaces de acceder a estos datos de una forma sencilla no serviría de nada. Para esta explicación vamos a utilizar un objeto Series.

data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd'])

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

data['b']

```
0.5
```

data['e'] = 1.25

```
a    0.25
b    0.50
c    0.75
d    1.00
e    1.25
dtype: float64
```

Acceder a estos datos y modificarlos es muy sencillo y flexible lo cual nos da una herramienta muy potente para trabajar con estos objetos anteriormente comentados.

Para seleccionar datos en un *Dataframe* tenemos múltiples formas. Por ejemplo, podemos obtener un objeto Series seleccionando una única columna de un *Dataframe*.

data['area']

```
California    423967
Florida       170312
Illinois      149995
New York      141297
Texas         695662
Name: area, dtype: int64
```

Para realizar una selección de columnas y obtener un *Dataframe* más pequeño a partir de nuestro *Dataframe* original podemos utilizar los métodos loc o iloc. Loc se puede utilizar junto con una lista con los nombres de las columnas o filas a seleccionar. Mientras que iloc se puede utilizar con números, para seleccionar las columnas o filas según su posición. Estos métodos se pueden combinar utilizando ix el cual nos ofrece una aproximación hibrida de estos.

El método loc nos ofrece el ser capaces de seleccionar los valores según un patrón boolean.

data.loc[data['density'] > 100, ['pop', 'density']]

De esta forma podemos simular consultas como si de una tabla SQL se tratara. Esto se tratará de una forma más detallada.

8.1.4 Comparación con SQL

Debido a que se lleva utilizando mucho tiempo Pandas tiene una gran cantidad de usuarios potenciales que estén acostumbrados a trabajar con SQL. Por tanto, muchas de las consultas que son posibles de realizar en SQL sobre una BD o varias puede realizarse igualmente sobre un *Dataframe* o varios. A continuación, voy a poner unos ejemplos muy sencillos para ver como se hace y así mostrar la potencia de la herramienta.

En mi caso esto era así, tenía conocimientos previos de SQL ya que se aprende durante el grado y además he hecho algún proyecto anterior donde lo he utilizado. Por tanto, a la hora de generar los resultados se puede pensar la consulta en SQL y posteriormente hacerla en Pandas.

Por ejemplo, para hacer un simple SELECT:

SELECT importe_total, propina, mesa, tiempo

FROM propinas

En Pandas se puede hacer como

propinas[importe_total, propina, mesa, tiempo]

Vamos a ver a continuación como hacer una consulta con un WHERE. Estas son las más útiles en un trabajo como este, ya que definir condiciones para realizar búsquedas nos permite obtener los resultados deseados.

SELECT *

FROM propinas

WHERE tiempo = ‘Comida’

Una consulta de este estilo en pandas se puede hacer como.

propinas[propinas[‘tiempo’] == ‘Comida’]

Lo que hace la sentencia anterior es pasar una Series de valores True/False al *Dataframe*, entonces se devuelven las filas donde los valores son True.

Evidentemente al igual que en SQL se pueden añadir los operadores OR y AND para poder pasar múltiples condiciones al *Dataframe*.

SELECT *

FROM propinas

WHERE tiempo = ‘Comida’ AND propina > 5.00

Y en Pandas.

propinas[(propinas[‘tiempo’] == ‘Comida’) & (propinas[‘propina’] > 5.00)]

El resto de las operaciones de SQL, como GROUP BY o los JOIN también se pueden realizar en Pandas. Pero estas no se van a comentar ya que no se han utilizado para nada en este trabajo.

8.1.5 Modificando los datos no disponibles

A diferencia de en la inmensa mayoría de los tutoriales o cursos relacionados con el *Data Science* los datos del mundo real son limpios y homogéneos. La mayoría de los *dataset* contienen datos con valores nulos. Por tanto, debemos hacer algo con el fin de poder obtener resultados válidos. Como se ha comentado anteriormente esto también es así con el *Dataframe* generado con la BD de ISBSG.

Aunque hay varias aproximaciones para tratar los valores nulos. En este trabajo se utiliza la de deshacernos de los datos nulos. Por tanto, utilizamos la función dropna()[20], la cual tiene una serie de parámetros que vamos a modificar para que haga lo que nosotros queremos. Por ejemplo, podemos definir que la función actue sobre las columnas del *Dataframe* en lugar de sobre las filas. Si esto lo combinamos con un margen, podemos eliminar las columnas que tienen un % de nulls mayor al que nosotros pongamos. Para hacer esto utilizamos el parámetro thresh el cual nos permite especificar el el numero de valores nulos que puede haber en la columna para que nos la "quedemos".

```
df.dropna(axis=1, thresh=int(0.5*len(df)))
```

En el trabajo una vez hemos eliminado las columnas con mayor número de valores nulos. Si que eliminamos los proyectos o filas con cualquier valor nulo. De esta manera el *Dataframe* restante no contiene ningún valor null.

8.2 Select_features.py

Quiero mencionar la librería select_features.py la cual está desarrollada desde 0 por mi para este proyecto. Dado que el proyecto forma de tres partes claramente diferenciadas, preparación de los datos, cálculo de resultados y representación de estos, a nivel de código también se divide en tres partes.

La primera como se ha comentado anteriormente se produce en un jupyter notebook lo que facilita el desarrollo de esta. Ocurre lo mismo con la representación de los resultados.

Pero para generar los cálculos lo más sencillo era generar una librería a parte con los métodos necesarios para realizar los cálculos. Tener el código separado nos permite reutilizarlo en caso necesario en otros proyectos de Python. Por tanto, instalando las dependencias necesarias de los módulos utilizados en los cálculos tenemos una librería totalmente funcional. Todo el código se ha desarrollado teniendo en mente su posible reutilización de cara al futuro y este se publicará en github. Aun así, esto no ha sido posible al 100% ya que hay funciones que se han creado específicamente para este proyecto y por tanto no se podrán reutilizar en otros. Por ejemplo, para utilizar las funciones en R se tiene que recodificar el *dataframe* para que sea un objeto en R válido.

Las funciones disponibles en esta librería son las siguientes:

- setupR_enviroment

[20] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.*Dataframe*.dropna.html

- setupR_enviroment_knn
- calcular_mi_R
- calcular_mi_R_2v
- calcular_mrmr_R
- recode_*dataframe*_R
- calc_mi_scikit
- calcular_mi_manual
- calcular_mi
- recode_*dataframe*
- recode_*dataframe*_v2
- calcular_mrmr_v2
- calcular_mmre
- calcular_mmre_R
- determinar_numero_variables
- evaluator_r
- evaluator
- greedy_forward_selection
- greedy_forward_selection_r
- doquire_forward_selection
- doquire_forward_selection_r

Como se puede intuir por los nombres de las funciones se han creado hasta cinco diferentes para realizar el cálculo de MI para distintos usos. Entre los usos están evidentemente el hecho de comprobar los resultados de las funciones comparándolos con otras.

Para comprobar resultados se han utilizado llamadas a métodos en R que realizan los mismos cálculos desde Python. Es por eso por lo que para todas las funciones en Python existen sus respectivas funciones en R.

```
def calcular_mi(variable, df):
    """Calcula Mutual Information para las variables independientes utilizando sklearn.feature_selection.mutual_info_regression para un DataFrame
    FUNCIÓN SOLO VALIDA PARA EL DATAFRAME DEL TRABAJO ORIGINAL

    Parameters:
        variable (String): Variable sobre la cual se calcula mi
        df (pandas.DataFrame): Dataframe

    Returns:
        pandas.Series: Serie con los resultados de mi
    """
    variables = ['Industry Sector', 'Application Group', 'Development Type', 'Development Platform', 'Language Type', 'Primary Programming Language',
                 'Functional Size', 'Adjusted Function Points', 'Project Elapsed Time', '1st Data Base System', 'Used Methodology']
    X = df.loc[:, variables]
    y = df.loc[:, variable].values
    mi = mutual_info_regression(X, y, n_neighbors=1)
    mi = pd.Series(mi)
    mi.index = X.columns
    mi = mi.sort_values(ascending=False)
    return mi
```

Figura 24 Función de cálculo de MI con mutual_info_regression

```python
def calcular_mmre_R(variable_a_imputar, df, k=5):
    if utils == None:
        setupR_enviroment()
    total = len(df)
    resultado = pd.DataFrame(columns=['Valor Original', 'Valor Imputado'])
    numero_columna = df.columns.get_loc(variable_a_imputar)
    pandas2ri.activate()
    for i in range(total):
        df_test = df.copy(deep=True)
        dato_original = df_test[variable_a_imputar].iloc[i]
        df_test[variable_a_imputar].iloc[i] = np.nan
        r_df_test = ro.conversion.py2rpy(df_test)
        r_df_test_imputed = knn_r(
            r_df_test, variable=variable_a_imputar, numFun=media, k=k)
        dato_imputado = r_df_test_imputed[variable_a_imputar].iloc[i]
        resultado = resultado.append(
            {'Valor Original': dato_original, 'Valor Imputado': dato_imputado}, ignore_index=True)
    mmre = sum(abs(resultado['Valor Original'] -
                   resultado['Valor Imputado'])/resultado['Valor Original'])/total
    return mmre, resultado
```

Figura 25 Función de cálculo mmre en R

Pese a esta librería, no todas las funciones están implementadas en ella, funciones cuya utilidad es exclusiva de este proyecto se han extraído de esta para evitar ruido y hacer un código más mantenible y legible. Por ejemplo, las funciones utilizadas para modificar el *dataframe* y hacerlo válido en R. Los nombres de las columnas del *dataframe* no puede tener espacios en R mientras que en Python no es un problema. Como solución sencilla se pueden cambiar los espacios por "_".

8.3 Dimensión de la solución propuesta

En total el proyecto ocupa 33.5MB incluyendo a las BDs. Esto se debe al tamaño de las BDs y al de los archivos jupyter notebook generados durante el desarrollo y para la representación de resultados.

A continuación, se explican en detalle la carpeta con el código más importante del proyecto.

Los archivos con código Python, es decir los ".py" son cuatro y tienen un total de 708 líneas de código y 126 de comentarios. Se ha de tener en cuenta que Python es un lenguaje de programación donde muchas operaciones que en otros lenguajes ocupan unas cuantas líneas en Python se realizan en una. Por tanto la generación de ese código es más rápida que en otros lenguajes.

Java

```java
public class Main {
  public static void main(String[] args) {
      System.out.println("hello world");
    }
}
```

Python

```python
print("hello world");
```

Ilustración 4 Comparación código en Java y Python

```
D:\Users\ivana\Documents\TFG>cloc-1.86.exe src
      23 text files.
      23 unique files.
      11 files ignored.

github.com/AlDanial/cloc v 1.86  T=0.88 s (17.0 files/s, 72781.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
HTML                             4            290             71          60927
Python                           4            113            126            708
Jupyter Notebook                 7              0           1475            389
-------------------------------------------------------------------------------
SUM:                            15            403           1672          62024
-------------------------------------------------------------------------------
```

Figura 26 datos del código del proyecto

Siguiendo las guías de PEP 257 se han generado los “docstrings” de los métodos de la librería select_features.py. Así como se ha añadido algún comentario para alguna línea más compleja.

```
def calcular_mmre(variable, df, k=5):
    """Calcula mmre imputando los valores con la funcion sklearn.impute.KNNImputer

    Parameters:
        variable (String): Variable sobre la cual se va a calcular mmre
        df (pandas.DataFrame): DataFrame
        k (int): Valor utilizado en n_neighbors de KNNImputer

    Returns:
        float con el valor de mmre
    """
```

Figura 27 ejemplo docstring para la función calcular_mmre

8.4 Distintas implementaciones de MI

Esta primera gráfica se corresponde a la librería info_gain.

https://github.com/Thijsvanede/info_gain

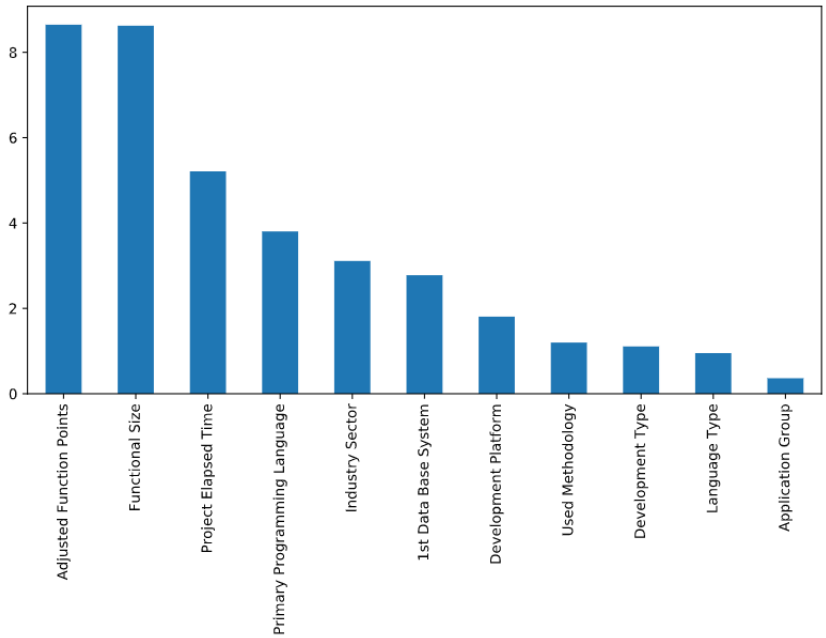

Figura 28 MI con info_gain

Esta grafica se hace aplicando manualmente al *dataframe* mutual_info_score de scikit

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mutual_info_score.html

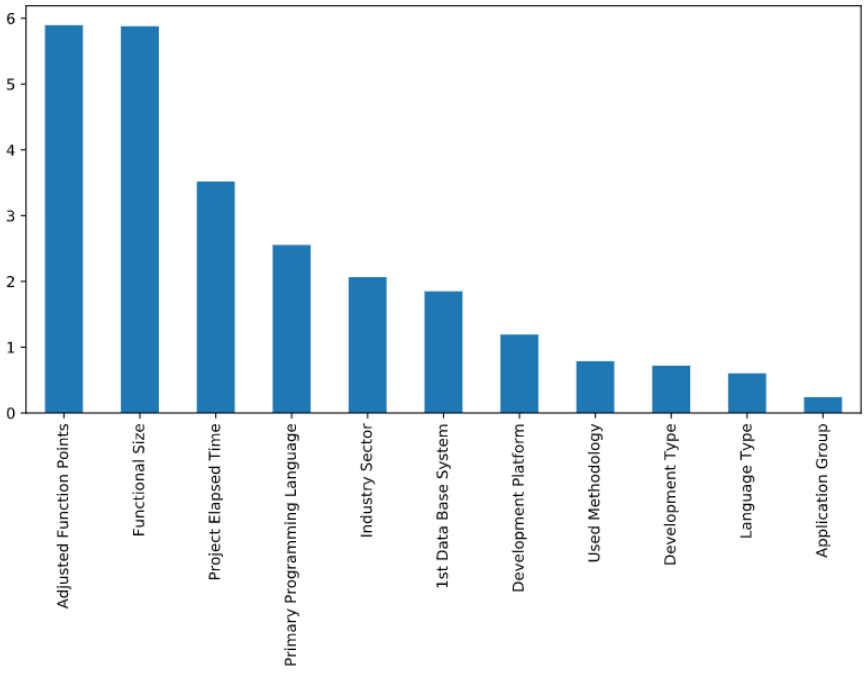

Figura 29 MI con mutual_info_score

La tercera gráfica corresponde al método que comentamos en la reunión mutual_info_regression.

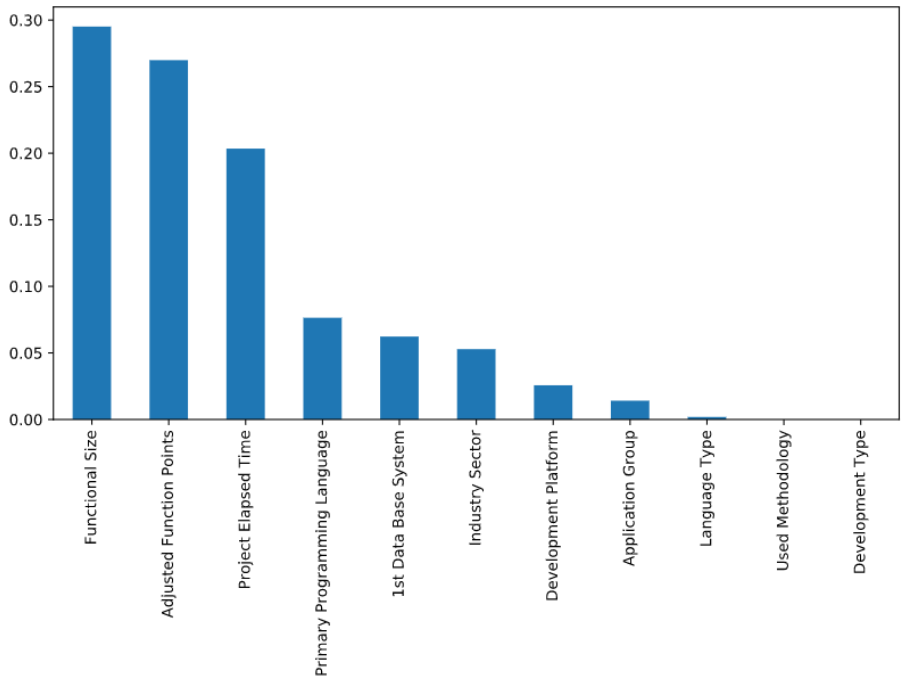

Figura 30 MI con mutual_info_regression

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_regression.html

Gráfica de Mi con Fselector

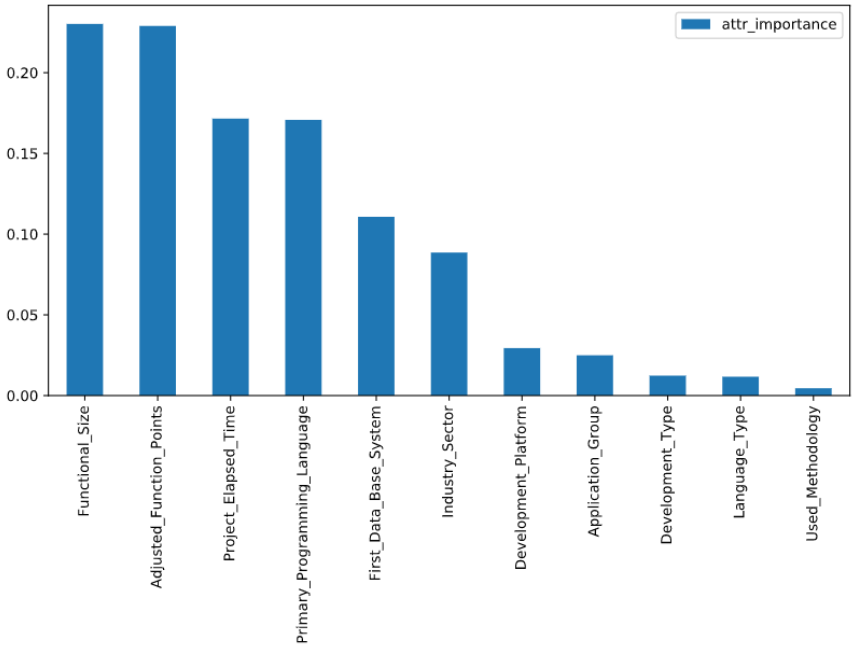

Figura 31 MI con FSelector

https://cran.r-project.org/web/packages/FSelector/index.html

8.5 Integración de Python con R

A pesar de las múltiples herramientas que tiene Python para realizar los cálculos de MI, se decide realizar una integración con el paquete FSelector de R. Para realizar la misma se utiliza el módulo RPY2[21].

Para realizar la instalación del módulo rpy2, se utiliza el procedimiento habitual.

"*pip install rpy2*"

Para poder utilizar RPY2 correctamente es necesaria la iniciación del entorno con los paquetes necesarios.

```
utils = rpackages.importr('utils')

utils.chooseCRANmirror(ind=1)

packages = ('FSelector')

utils.install_packages(StrVector(packages))

FSelector = importr("FSelector")

information_gain = FSelector.information_gain
```

Si es cierto que no todo es tan fácil ya que por lo menos trabajando desde Windows debemos tener configurado correctamente el PATH. Así como el entorno Java correcto en caso de tener R instalado en 64 bits Java tiene que ser también de 64 bits. En mi ordenador estaba instalado por defecto la versión de 32bits, a pesar de que el sistema operativo es de 64.

La integración de R en Python nos permite combinar las ventajas de ambos lenguajes en un entorno único y más productivo. Además de poder utilizar paquetes de R para los que no existe un equivalente nativo en Python.

Recalcar que al utilizar esta librería si modificamos la asignación de recursos del sistema operativo, en este caso Windows obtenemos una mejora notable en los tiempos de ejecución. Evidentemente la utilización de la librería ralentiza el proceso respecto a realizarse de forma nativa en cualquiera de los dos lenguajes.

8.6 Multiprocesamiento en Python [XIII]

Lo primero que es necesario definir es la diferencia entre el multiprocesamiento y la ejecución multihilo. En el multiprocesamiento múltiples procesos se ejecutan utilizando uno o más núcleos. Mientras que en la ejecución multihilo, múltiples hilos se ejecutan sobre un mismo proceso. Por tanto, la ejecución multihilo nos viene bien para tareas que se bloquean por la I/O o cuando se espera a una respuesta o comunicación en red, etc.

[21] https://pypi.org/project/rpy2/

Cuando lo que necesitamos es obtener más potencia de las cpus para realizar cálculos complejos utilizaremos el multi procesamiento.

Aclarado esto a la hora de realizar este trabajo se ha investigado con las librerías de Python de multiprocesamiento con el fin de acelerar la ejecución.

Con la ayuda de la documentación oficial de Python se puede consultar información sobre el módulo multiprocessing[22]. Una vez con esto se empieza a hacer pruebas generando múltiples procesos.

```
from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

El problema de este método es que debemos generar los procesos, iniciarlos y después asegurarnos de que estos finalizan uno a uno. Esto se puede realizar utilizando una lista de procesos e iterando a través de esta. Pero esto nos deja el problema de que necesitamos muchas iteraciones ya que nuestro script realiza 500 runs por cada algoritmo, para cada valor de k. Por tanto, generar 8000 procesos no es una opción real. Por lo menos no en un ordenador de uso doméstico.

Por lo tanto, buscando en la documentación llegamos al módulo Pool. Este crea una "piscina" de procesos donde definimos un número máximo de procesos o trabajadores y conforme se van utilizando y terminando estos vuelven a ser lanzados hasta completar todos los cálculos.

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

Esto se fue desarrollando mientras el ordenador iba procesando iteraciones del script de resultados. Debido a que a pesar de tener resuelto el problema de ir generando los procesos e iniciarlos conforme van terminando nos queda el hecho de que se han de

[22] https://docs.python.org/3/library/multiprocessing.html

recoger los resultados. Para cuando se obtiene una solución correcta del script de resultados utilizando multiprocesamiento los resultados ya se han generado.

Para intercambiar datos entre procesos se pueden utilizar Pipes[23], de esta forma se pueden enviar datos entre el proceso principal y los trabajadores.

Realmente la solución se puede realizar, pero ya no es útil ya que se han obtenido los datos de forma secuencial antes de tener terminado el script concurrente.

8.7 Graficas.ipynb

Como es evidente el primer paso a realizar es la importación de todas las dependencias.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
datak1 = pd.read_csv("500 cross k1 v2.csv", sep=';')
sns.set() #Define Seaborn como herramienta de PLOT por defecto
```

Separación de los datos dependiendo del algoritmo de origen. (MI_1L, MI_2L, mRMR_1L, mRMR_2L)

In [ ]:

```
filtro1 = (datak1['Metodo'] == 1)
filtro2 = (datak1['Metodo'] == 2)
filtro3 = (datak1['Metodo'] == 3)
filtro4 = (datak1['Metodo'] == 4)
data1k1 = datak1.loc[filtro1,:]
data2k1 = datak1.loc[filtro2,:]
data3k1 = datak1.loc[filtro3,:]
data4k1 = datak1.loc[filtro4,:]
```

Cálculo de las medias acumuladas de los valores de MMRE para los distintos algoritmos. Y generación del diagrama de líneas.

In [ ]:

```
data_mean1 = data1k1['MMRE'].expanding().mean()
data_mean2 = data2k1['MMRE'].expanding().mean()
data_mean2.index = range(500)
data_mean3 = data3k1['MMRE'].expanding().mean()
data_mean3.index = range(500)
data_mean4 = data4k1['MMRE'].expanding().mean()
data_mean4.index = range(500)

datafig2_1 = pd.Dataframe({
'MI_1L':data_mean1,
'mRMR_1L':data_mean2,
'MI_2L':data_mean3,
'mRMR_2L':data_mean4})
```

Calculo de las medias acumuladas de los valores de MMRE para los distintos algoritmos para los proyectos ágiles. Y generación del diagrama de lineas

[23] https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Pipe

In [ ]:

```
datak1 = pd.read_csv("Datos Agile.csv", sep=';', decimal=",")
metodo1 = datak1['k1']
metodo2 = datak1['k2']
metodo3 = datak1['k3']
metodo4 = datak1['k4']
data_meanA1 = metodo1.expanding().mean()
data_meanA2 = metodo2.expanding().mean()
data_meanA3 = metodo3.expanding().mean()
data_meanA4 = metodo4.expanding().mean()

datafig2a_1 = pd.Dataframe({
'MI_1L':data_meanA1,
'mRMR_1L':data_meanA2,
'MI_2L':data_meanA3,
'mRMR_2L':data_meanA4})
```

Figura con los 2 diagramas de líneas, proyectos tradicionales y ágiles.

In [ ]:

```
sns.set_context("paper")
fig, axs = plt.subplots(nrows=2, figsize=(16,16))
datafig2_1.plot(ax=axs[0])
axs[0].set_title('Tradicionales')
datafig2a_1.plot(ax=axs[1])
axs[1].set_title('Ágiles')
```

Generación del Boxplot para los valores de MMRE

In [ ]:

```
plt.figure(figsize=(15,15))
sns.boxplot(data=[data1k1['MMRE'], data2k1['MMRE'], data3k1['MMRE'], d
ata4k1['MMRE']])
```

Generación del *Dataframe* de los datos para la tabla 5 del trabajo.

In [ ]:

```
#Tabla 5
data_table4 = {'MI_1L':[data1k1['MMRE'].mean(), data1k2['MMRE'].mean()
, data1k3['MMRE'].mean(), data1k4['MMRE'].mean()], 'mRMR_1L':[data2k1[
'MMRE'].mean(), data2k2['MMRE'].mean(), data2k3['MMRE'].mean(), data2k
4['MMRE'].mean()], 'MI_2L':[data3k1['MMRE'].mean(), data3k2['MMRE'].me
an(), data3k3['MMRE'].mean(), data3k4['MMRE'].mean()], 'mRMR_2L':[data
4k1['MMRE'].mean(), data4k2['MMRE'].mean(), data4k3['MMRE'].mean(), da
ta4k4['MMRE'].mean()]}
table4df = pd.Dataframe(data_table4, index=['K = 1', 'K = 2', 'K = 3',
 'K = 4'])
table4df
```

Generación de la tabla de tiempos para los distintos algoritmos del trabajo.

In [ ]:

```
#Table 5
data_table5 = {'Media':[data1k1['Tiempo'].mean(), data2k1['Tiempo'].me
an(), data3k1['Tiempo'].mean(), data4k1['Tiempo'].mean()], 'Desviación
```

```
 Típica':[data1k1['Tiempo'].std(), data2k1['Tiempo'].std(), data3k1['T
iempo'].std(), data4k1['Tiempo'].std()]}
table5df = pd.Dataframe(data_table5, index=['MI_1L', 'mRMR_1L', 'MI_2L
', 'mRMR_2L'])
table5df
```

Re-Codificación de las variables para que sean tratadas como variables categóricas, en lugar de ser tratadas como strings.

In [ ]:

```
data_variables = data1k1['Variables Elegidas']
final1 = list()
for item in data_variables.iteritems():
    aux = list(item[1].split(', '))
    linea = list()
    for word in aux:
        word = word.replace('[', '')
        word = word.replace(']', '')
        word = word.replace("'", '')
        linea.append(word)
    final1.append(linea)

data_variables = data2k1['Variables Elegidas']
final2 = list()
for item in data_variables.iteritems():
    aux = list(item[1].split(', '))
    linea = list()
    for word in aux:
        word = word.replace('[', '')
        word = word.replace(']', '')
        word = word.replace("'", '')
        linea.append(word)
    final2.append(linea)

data_variables = data3k1['Variables Elegidas']
final3 = list()
for item in data_variables.iteritems():
    aux = list(item[1].split(', '))
    linea = list()
    for word in aux:
        word = word.replace('[', '')
        word = word.replace(']', '')
        word = word.replace("'", '')
        linea.append(word)
    final3.append(linea)

data_variables = data4k1['Variables Elegidas']
final4 = list()
for item in data_variables.iteritems():
    aux = list(item[1].split(', '))
```

```
    linea = list()
    for word in aux:
        word = word.replace('[', '')
        word = word.replace(']', '')
        word = word.replace("'", '')
        linea.append(word)
    final4.append(linea)
```

Generación de los countplots del número de variables seleccionadas por cada algoritmo.

In [ ]:

```
x1 = list()
for fila in final1:
    x1.append(len(fila))

x2 = list()
for fila in final2:
    x2.append(len(fila))

x3 = list()
for fila in final3:
    x3.append(len(fila))

x4 = list()
for fila in final4:
    x4.append(len(fila))

fig, axs = plt.subplots(2, 2, figsize=(12,12))
sns.countplot(x1, ax=axs[0, 0])
axs[0, 0].set_title('MI_1L')
sns.countplot(x2, ax=axs[0, 1])
axs[0, 1].set_title('mRMR_1L')
sns.countplot(x3, ax=axs[1, 0])
axs[1, 0].set_title('MI_2L')
sns.countplot(x4, ax=axs[1, 1])
axs[1, 1].set_title('mRMR_2L')
```

Generación de los datos del recuento de uso de cada variable, Tabla 8

In [ ]:

```
#datos tabla 7
variablesdf['Functional Size'].value_counts()
listavariables = ['Adjusted Function Points', 'Functional Size', 'Proj
ect Elapsed Time', 'Primary Programming Language', 'Industry Sector',
'1st Data Base System', 'Development Platform', 'Used Methodology', 'D
evelopment Type', 'Language Type', 'Application Group']
for var in listavariables:
    print(variablesdf[var].value_counts())
```

8.8 Import_agile.ipynb

```
datak1T = pd.read_csv("500 cross k1 v2.csv", sep=';')
```

```
datak1 = pd.read_csv("Datos Agile.csv", sep=';', decimal=",")
filtro1 = (datak1T['Metodo'] == 1)
filtro2 = (datak1T['Metodo'] == 2)
filtro3 = (datak1T['Metodo'] == 3)
filtro4 = (datak1T['Metodo'] == 4)
data1k1 = datak1T.loc[filtro1,:]
data2k1 = datak1T.loc[filtro2,:]
data3k1 = datak1T.loc[filtro3,:]
data4k1 = datak1T.loc[filtro4,:]
```

In [ ]:

```
metodo1 = datak1['k1']
metodo2 = datak1['k2']
metodo3 = datak1['k3']
metodo4 = datak1['k4']
data_mean1A = metodo1.expanding().mean()
data_mean2A = metodo2.expanding().mean()
data_mean3A = metodo3.expanding().mean()
data_mean4A = metodo4.expanding().mean()

datafig2_1A = pd.Dataframe({
'MI_1L':data_mean1A,
'mRMR_1L':data_mean2A,
'MI_2L':data_mean3A,
'mRMR_2L':data_mean4A})
datafig2_1A.plot(figsize=(16, 8))
```

In [ ]:

```
data_mean1 = data1k1['MMRE'].expanding().mean()
data_mean2 = data2k1['MMRE'].expanding().mean()
data_mean2.index = range(500)
data_mean3 = data3k1['MMRE'].expanding().mean()
data_mean3.index = range(500)
data_mean4 = data4k1['MMRE'].expanding().mean()
data_mean4.index = range(500)

datafig2_1 = pd.Dataframe({
'MI_1L':data_mean1,
'mRMR_1L':data_mean2,
'MI_2L':data_mean3,
'mRMR_2L':data_mean4})
datafig2_1.plot(figsize=(16, 8))
```

In [ ]:

```
#Figura MI_1L
datafigMI_1L = pd.Dataframe({
    'Agiles': data_mean1A,
    'Tradicionales' : data_mean1
})
datafigMI_1L.plot(figsize=(16, 8))
```

In [ ]:

```
plt.figure(figsize=(15,15))
dataBoxAgil = pd.Dataframe({
    'MI_1L':metodo1,
    'mRMR_1L':metodo2,
    'MI_2L':metodo3,
    'mRMR_2L':metodo4
})
sns.boxplot(data=dataBoxAgil)
```