ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO INGENIERIA DEL SOFTWARE + MATEMÁTICAS

Curso Académico 2021/2022

Trabajo Fin de Grado

DESARROLLO E INTERFAZ DEL ESTUDIO DE DIVERSIFICACIÓN DE INVERSIONES SOBRE IBEX-35 UTILIZANDO CLUSTERING

Autor:
Javier Méndez García-Brioles
Directores:
Regino Criado

Indice

1 Resumen

Este proyecto tiene como finalidad explicar el desarrollo de los algoritmos e interfaz que permiten a [1] proponer un plan de inversión alternativo para minimizar la volatilidad de la inversión, y por tanto, minimizar el riesgo.
El plan de inversión consiste en aplicar una modificación del algoritmo k-means para agrupar los stocks en k clusters y de esos clusters sacar un optimo, dando lugar a k stocks diferentes en los que invertir.
Con esté plan haremos un estudio sobre los stoks del IBEX-35 comparando nuestro algoritmo con planes de inversión convencionales para ver como se consigue la mínima volatilidad. Ademas haremos un estudio de previsión para ver si nuestro algoritmo mantiene la mínima volatilidad si añadimos datos posteriores a los que tiene como input el algoritmo.
Además facilitaremos una interfaz gráfica para mostrar e interactuar con los datos.

2 Objetivos

Desde siempre ha existido la necesidad de comparar datos y ver su relación entre ellos, y una muy buena forma de ejercer esta comparación es clasificando los estos datos en conjuntos de datos. Dentro de cada conjunto los datos tienen mayor relación entre ellos que con los datos de fuera del conjunto, pero la búsqueda de estos conjuntos no es fácil. En la actualidad a esta búsqueda se la denota como clustering, que es puede considerar como el proceso por el cual se crean conjuntos de elementos similares. Este proceso se puede aplicar sobre todo conjunto de datos sobre los que se pueda aplicar una comparación entre ellos pero en este caso nos centraremos en buscar clusters de ondas temporales mediante varios métodos que comentaremos a continuación.
En un principio nos íbamos a basar en el grafo de visibilidad sacado de cada stock para, mediante sus propiedades, agrupar estos stocks. Pero al decidirnos por agrupar los stocks mediante k-means, la transformación a grafo de visibilidad ha pasado a un segundo plano, sirviendo como medio para transformar nuestras series iniciales a otras series que recojan las propiedades de los grafos de visibilidad.
Partimos de un conjunto de elementos, los cuales están compuestos por una sucesión de valores a lo largo de un intervalo de tiempo, estos elementos se pueden clasificar tal cual, o se puede hacer una transformación antes de su clasificación y utilizar esa transformación como input. En nuestro caso exploraremos 3 caminos:
1 Realizamos la clasificación sobre los elementos sin transformar.
2 Transformamos los elementos en una versión simplificada de ellos mismos para intentar mejorar el tiempo de ejecución.
3 Transformamos los elementos en su correspondiente grafo de Visibilidad y luego aplicamos el algoritmo sobre esos grafos.
Este proyecto se basara en la clasificación de series temporales mediante una variación sobre un método muy conocido llamado K-Means, lo que nos permite clasificar elementos siempre que se pude definir una función distancia entre ellos.
Una vez que tenemos nuestros elementos separados en distintos conjuntos, después de hacer las transformaciones y aplicar el algoritmo K-means tenemos k conjuntos cuyos elementos son similares entre ellos, esto resulta muy útil ya que nuestros elementos de entrada son stocks, por lo tanto saber que stocks se parecen entre ellos nos permite evitar elegir stocks que se comportan de forma parecida lo que nos permite minimizar la volatilidad de la

inversión. Por último, una vez llegado al resultado que es la volatilidad de la inversión podremos comparar todos los caminos que hemos utilizado entre ellos y con resultados de control (como son invertir de forma homogénea en todos los stocks, invertir todo en el stock menos volátil o invertir en n stocks aleatorios) para ver así que variación del método es el mejor o si los resultados de control son superiores.
Durante todo el proyecto trabajaremos sobre los datos producidos por las empresas dentro del IBEX-35 y proporcionados por una API de yahoo finances.
Desde un punto de vista más teórico nos centraremos en evolucionar el algoritmo K-means para que se adecue a nuestras necesidades, lo que consiste en hacer modificaciones para que el K-means acepte distancias un poco menos convencionales que la genérica, ejecutar el K-means varias veces para mejorar la precisión de los conjuntos y por ultimo modificar K-means para que también funcione sobre un grafo.

3 Fundamentos Teóricos

3.1 Ondas Temporales

Ondas Simples
Llamaremos ondas simples a las cuales compuestas exclusivamente por una función seno, con sus posibles modificaciones de amplitud, frecuencia, y desplazamiento.
Ondas Compuestas
Llamaremos ondas compuestas a la superposición de ondas simples a lo largo del tiempo
Transformada de Fourier
La transformada de Fourier es una función matemática que permite relacionar un dominio temporal con un dominio de frecuencias en ambos sentidos
Transformada Wavelet
La transformada Wavelet es una función muy utilizada en compresión de imágenes que permite dividir la cantidad de valores manteniendo la mayor

cantidad posible de detalle.

3.2 Teoría de Grafos

Grafo
Un grafo ($G = (V, E)$) es una estructura de datos compuesta por vértices (V), donde se almacenan los datos, y aristas o arcos ($E = V \times V$), que representan las conexiones entre dos vértices.
Grafo no Dirigido
Grafo ($G = (V, E)$) tal que:

$$V \neq \emptyset$$

$$E \subseteq \{(a, b)/(a, b) \in E \rightarrow (b, a) \in E\}$$

Vértice
Componente mínimo indivisible por el que están compuestos los grafos
Arista
Componente de un grafo que se caracteriza por conectar dos vértices (E)
Grado
El grado de un vértice ($v \in V$) es:

$$degre(v) = |E_v|$$

$$E_v \subseteq \{(a, b) \in E/a = v \vee b = v\}$$

Orden
El orden de un grafo ($G = (V, E)$) es:

$$orden(G) = degre(w)$$

con

$$w \in V/\forall v \in V degre(w) \geq degre(v)$$

Grafo de Visibilidad
Un grafo de visibilidad es el grafo asociado a una onda temporal siguiendo el siguiente proceso:
-A cada valor de la onda se le asigna un nodo del grafo.
-Las conexiones se deciden según que nodos se ven entre ellos sin que otro nodo les bloque la visión. Esta restricción equivale a cumplir la ecuación ($y_m + \frac{y_n - y_m}{n-m}(k - m) \geq y_k$) para todo nodo k entre m y n ($m < k < n$)[8]

3.3 Centralidad

La centralidad en un grafo consiste en la asignación de un número a cada nodo del grafo, este valor es sumamente importante ya que permite comparar nodos entre sí.
La importancia de la centralidad reside en que encontrando nodos con valores muy altos de centralidad es equivalente a encontrar los nodos más importantes del grafo, los cambios a estos nodos causarán el mayor impacto dentro del grafo. Este valor es externo al propio nodo y puede variar con cualquier modificación del grafo, aunque esa modificación no afecte al grafo en sí.
La centralidad es de suma importancia pues nos permite estudiar las conexiones entre objetos, ya sea entre personas a través de medios como las redes sociales, o las conexiones por carretera entre ciudades. Y este estudio por ejemplo nos permite saber la influencia de cierta persona x, o el trafico probable en una ciudad.
Hay varias medidas que se pueden usar para determinar la centralidad, pero nosotros nos vamos a centrar en la centralidad de grado y la interpolación(Betweenness)
Centralidad de grado
La centralidad de grafo es posiblemente la medida de centralidad más fácil de entender. Consiste en contar el número de conexiones que tiene cada nodo, lo cual nos basta para ver si un nodo esta incomunicado o cual es la distancia entre cualquier pareja de nodos del grafo. Estos son datos muy útiles para el estudio de grafos, pero para ciertas aplicaciones se nos quedan cortos.
Intermediación
La intermediación es un poco más compleja y consiste en calcular el numero de veces que un nodo se utiliza como puente en la ruta más corta entre una pareja de nodos. Pongamos el ejemplo de una red de ordenadores, el nodo con una intermediación más alta seria por el cual pasa más información, sabiendo esto podemos ver que si ese nodo se incapacita sería el que más problemas ocasionaría a la red. Por lo tanto, con esta información podríamos saber los nodos que necesitan más protección en todo momento, esto es algo que con un simple estudio del grado de cada nodo no se vería.

3.4 Distancias

Una distancia es cualquier función que cumple lo siguiente:

1. $d(x, y) = 0 \iff x = y$

2. $d(x, y) = d(y, x)$ (simetría)

3. $d(x, z) \leq d(x, y) + d(y, z)$ (desigualdad triangular)

Lo que nos dice de forma análoga por la desigualdad triangular:
$d(x, y) \geq 0$ (no negatividad)

3.5 Volatilidad

La volatilidad es un termino que mide la variación de los valores a lo largo del tiempo, a mayor variación mayor volatilidad. Nuestra definición especifica de volatilidad va a ser bastante estándar.
Definimos volatilidad como:

$$Volatilidad = \sum_{i=1}^{n} \sqrt{|valor_i - media|}$$

3.6 Complejidad Computacional

Toda la teoría de Complejidad Computacional consiste en hacernos la pregunta, ¿que tareas son fáciles y que tareas son difíciles para un ordenador?. Al hacernos esta pregunta nos damos cuenta de no hay forma fácil de responder con lógica convencional ya que una tarea fácil para un ser humano no equivale a una tarea fácil para un ordenador. Pongamos como ejemplo reconocimiento de imágenes, si tenemos una imagen de un perro y se la enseñamos a una persona sabrá al momento que es un perro aunque no haya visto nunca esa foto exacta.
Pero para un ordenador eso resultaría imposible sin un gran trabajo por detrás y aunque en la actualidad ya existen algoritmos que permitan responder esa pregunta no están ni de lejos a la altura de una persona.
A su vez si tenemos cualquier tarea de computación pura un ordenador tardará mucho menos que una persona. Así que resumiendo hay tareas fáciles para personas que nos casi imposibles para ordenadores y tareas fáciles para ordenadores que son casi imposibles para personas.
Por lo tanto para llegar a alguna conclusión vamos a necesitar una estrategia precisa que nos facilite algún indicador de que tareas son fáciles para un ordenador y eso es lo que hace el estudio de la Complejidad Computacional.
El estudio de la Complejidad se basa principalmente en estudiar dos parámetros, la cantidad de memoria utilizada por un algoritmo, y el numero de instrucciones que se ejecutan, esto equivale a medir el espacio y tiempo que gasta

un algoritmo. Nosotros solo nos vamos a centrar en el estudio del tiempo.

Para un algoritmo concreto podemos definir la siguiente función T(n) a la cual le pasamos un input de tamaño n y nos devuelve el número de operaciones que realiza el algoritmo en el peor de los casos.

$$T : N \to R^+$$

Como un ordenador realiza estas operaciones muy rápido los errores de magnitud pequeña pueden ser ignorados haciendo que solo nos importe el comportamiento asintótico de la ecuación conforme n crezca. Para expresar esto utilizaremos las siguientes definiciones.
Sean $T, S : N \to R^+$ se dice que T(n) **domina** a S(n) si existen $k > 0$ y n_o tal que para todo $n > n_0$:

$$S(n) \leq kT(n)$$

Con O(T(n)) siendo el conjunto de funciones dominadas por T(n) podemos decir que dos funciones T(n) y S(n) son **del mismo orden** si $S(n) \in O(T(n))yT(n) \in O(S(n))$. Se llama **complejidad de un algoritmo** al mayor orden O(n) al que pertenece T(n).
Dependiendo de cual sea el mayor O(n) al que pertenece T(n) podemos hablar de distintas complejidades.

1. Con $T(n) \in O(log(n))$ se dice que el algoritmo tiene **complejidad logarítmica**
2. Con $T(n) \in O(n)$ se dice que el algoritmo tiene **complejidad lineal**
3. Con $T(n) \in O(n^2)$ se dice que el algoritmo tiene **complejidad cuadrática**
4. Con $T(n) \in O(n^x)$ se dice que el algoritmo tiene **complejidad polinomial**
5. Con $T(n) \in O(e^n)$ se dice que el algoritmo tiene **complejidad exponencial**

3.7 Algoritmos

Algoritmo de Dijkstra
También conocido como algoritmo de mínimos caminos nos permite determinar el camino más corto entre un nodo origen y el resto de los nodos del grafo, si se hace este algoritmo sobre todos los nodos nos queda como resultado la mínima distancia entre todos los pares de nodos. Aquí se muestra su

pseudocódigo:

Algorithm 1 Algoritmo de Dijkstra

Input: Grafo $G = (V, E)$
Nodo inicial *node_init*
Output: Distancias $\mathcal{D} = \{d_1..d_n\}$

procedure DIJKSTRA($G, node\_init$)
$\mathcal{D} = \{d_1..d_n\} \leftarrow \{INF..INF\}$
$pred \leftarrow \{\emptyset..\emptyset\}$
$seen \leftarrow \{False..False\}$
$D[node\_init] \leftarrow 0$
 # Utilizamos una cola para asegurar el recorrido total del grafo
Q.add(node_init,D[node_init])
while $Q \neq \emptyset$ **do**:
    # Extraer elemento de mínima distancia
    $node \leftarrow Q.min()$
    $seen[node] \leftarrow True$
    for $v \in adjacent(node)$ **do**
        # Añadimos distancia si es menor que la actual
        if (**not** seen[v] **and** $\mathcal{D}[v] > \mathcal{D}[node] + E[node, v]$) **then**
            $\mathcal{D}[v] \leftarrow \mathcal{D}[node] + E[node, v]$
            $pred[v] \leftarrow node$
            Q.add(v,D[v])
return D;

Este algoritmo tiene una complejidad computacional $\mathbf{O(n^2)}$ ya que para cada nodo se puede llegar a recorrer todos los nodos. Pero si queremos saber las distancias entre todos los nodos la complejidad computacional aumenta a $\mathbf{O(n^3)}$ pues hay que ejecutar el algoritmo de Dijkstra para cada nodo.

K-means

El K-means o K-medias es un método de clasificación no supervisada que permite la clasificación de n elementos en k grupos distintos, en cada uno de estos k grupos los elementos dentro se consideran más parecidos entre ellos que con los elementos de fuera. K-means se centra en resolver un problema de optimización en el cual se busca minimizar la suma total de la distancia de los elementos a su centroide más cercano:

$$\min_{C} \sum_{i=1}^{k} \sum_{x_j \in C_i} d(\sigma_i, x_j)$$

Donde $d(x, y)$ es la función distancia utilizada, $\{x_1..x_n\}$ son los elementos, $C = \{C_1..C_k\}$ son los k clusters y $\{\alpha_1..\alpha_k\}$ son los centroides correspondientes a cada cluster.
La principal ventaja del k-means es que es muy rápido, pero tiene también una gran desventaja que hay que tener en cuenta, los resultados obtenidos dependen enormemente de como se inicializan los centroides. Por esta razón no podemos asegurar un optimo global sino que nos tendremos que conformar con un optimo local.
K-means consta de 3 partes indispensables:

1. **Inicialización de centroides**: Se determinan las posiciones iniciales de los centroides, esto se puede hacer de muchas formas, todas ellas validas y todas ellas pueden dar soluciones ligeramente distintas. Podemos elegir por ejemplo los k primeros elementos como centroides o elegirlos de forma aleatoria.

2. **Relacionar cada elemento con un centroide**: Para cada elemento se calcula la distancia a todos los centroides y se relaciona con el centroide más cercano. Cada elemento se añade al cluster del centroide correspondiente.

3. **Actualizar centroides**: Se actualizan los centroides haciendo la media de todos los elementos de su cluster y ese valor es el nuevo centroide.

Los pasos 2 y 3 se repiten hasta que llegamos a nuestra condición de parada, que suele ser que la diferencia entre los centroides anteriores y los nuevos centroides sea menor a un valor umbral.

El proceso se puede ver con más detalle en el siguiente pseudocódigo:
Para este algoritmo es difícil estudiar su complejidad computacional ya que

Algorithm 2 K-means clustering

Input: Data $\mathcal{X} = \{x_1..x_n\}$
Numero de clusters k
Valor umbral ε

Output: Clusters $\mathcal{C} = \{C_1..C_k\}$

procedure K-MEANS($\mathcal{X}, k, \varepsilon$)
$\mathcal{C} = \{C_1..C_k\} \leftarrow \{\emptyset..\emptyset\}$
$\{\sigma_1..\sigma_k\} \leftarrow$ **InicializarCentroides**($\mathcal{X}, k$)
loop:
    # Distribución de elementos en clusters
    $C_i \leftarrow \{x_j / d(\sigma_i, x_j) \leq d(\sigma_h, x_j) \forall h \in \{1..k\}\} \forall i \in \{1..k\}$
    # Actualización de clusters
    $new\sigma_i \leftarrow \frac{\sum_{x_j \in C_i} x_j}{|C_i|} \forall i \in \{1..k\}$
    # Comprobar la condición de parada
    if $(\sum_{i=1}^{k} d(\sigma_i, new\sigma_i) \leq \varepsilon)$ **then**
        return $\mathcal{C}$
    $\{\sigma_1..\sigma_k\} \leftarrow \{new\sigma_1..new\sigma_k\}$
    goto *loop.*
close;

el camino para llegar a la condición de parada es difícil de definir, por eso mismo se le considera un problema computacionalmente difícil (NP-hard), se han demostrado cotas de su complejidad como $\mathbf{O(n^{34}k^{34}d^8log^4(n)/\sigma^6)}$ o $\mathbf{O(dn^4M^2)}$ para casos más simples, pero nosotros nos referiremos a su complejidad como $\mathbf{O_{K-means}}$ para poder compárala con la complejidad de las modificaciones al algoritmo

4 Herramientas

Para este proyecto crearemos una aplicación python mediante la cual interactuaremos con la funcionalidad implementada, para esta implementación utilizaremos las siguientes herramientas:

4.1 Anaconda

Anaconda es una plataforma que gestiona entornos python para facilitar la instalación y uso de librerías python. Anaconda se centra en facilitar la programación científica, lo que incluye ciencia de datos, aprendizaje maquina y procesado de datos. Esto lo consigue implementando otra forma de instalar librerías a través del comando conda que equivale a el pip normal de python pero se centra en prevenir conflictos en la instalación de librerías, lo que produce un entorno más resistente a fallos de compatibilidad de librerías que con python convencional.

4.2 Spyder

Spyder es el IDE (Entorno de desarrollo integrado) que utilizaremos para la mayor parte de la programación. Viene por defecto con una gran cantidad de librerías integradas que resultan muy útiles para el desarrollo de código científico, como puede ser numpy o pandas. Ademas como es un entorno de código abierto, es decir, que permite que cualquiera pueda cambiar y distribuir el código, lo que hace que tenga una gran variedad de extensiones muy útiles para el desarrollo software.

4.3 Extensiones Integradas en Spyder

Aquí mencionaremos las principales extensiones que se han utilizado en el desarrollo de este proyecto:
-Spyder-Unittest: Es una extensión sobre Spyder que permite la automatización de test unitarios sobre nuestro código, ademas esta funcionalidad viene con una interfaz propia que permite la ejecución de todos los tests implementados a la vez y así poder ver al momento cuales fallan y cuales pasan.

-Spyder-GitHub: Es una extensión que permite acceder a GitHub desde dentro del entorno de desarrollo Spyder, lo que proporciona una pequeña interfaz gráfica para hacer commits y ver el historial del repositorio, ademas de permitir el uso de los comandos git dentro del terminal de Spyder.
-Spyder-Code Análisis: Esta extensión no da información sobre la calidad del código, muestra tanto las diferencias con las convenciones de como programar en python como las warnings y las vulnerabilidades del código.
-Spyder-Profiler: Esta extensión simplemente permite ver como está distribuido el tiempo de ejecución de la aplicación. El tiempo de ejecución se separa por método, lo que permite ver que métodos están consumiendo más recursos y esto ayuda en el proceso de optimización del código.

4.4 Qt Designer

Qt Designer es una herramienta que facilita el diseño de aplicaciones python aportando entorno donde poder crear, ver y gestionar la interfaz de una aplicación python. Este interfaz que se crea aquí se pude convertir en código python, el cual a su vez puede ser modificado en cualquier entorno de desarrollo python para añadir funcionalidad real a la aplicación. Qt Designer permite el uso de la librería gráfica QT en python lo que nos da opciones útiles que no nos dan otras librerías de interfaces de python como puede ser tkinter.

4.5 Trello

Trello es una aplicación con acceso web que permite la administración de proyectos. Se basa en un modelo kanban para poder llevar un seguimiento de las tareas a realizar en el desarrollo de nuestra aplicación.

4.6 Github

Github es un sistema de control de versiones que permite tener el código subido a un repositorio mientras que los cambios se hacen en local. Esto si se hace bien ayuda mucho en el desarrollo del código porque nos permite compartimentar cada parte de la funcionalidad en un commit individual que explique lo que se ha implementado en el mismo. Esto permite tener un buen historial de desarrollo para mejorar la mantenibilidad del código y permite tener versiones de código anteriores en caso de que durante el desarrollo rompamos la versión que tenemos del código en local y no sepamos arreglarlo,

en este caso siempre podemos volver a la versión anterior del código para solucionar el problema.

4.7 Programa Para Diagramas UML

Este programa nos permite la creación de diagramas UML (lenguaje unificado de modelado) para poder mostrar una gran cantidad de relaciones distintas, por ejemplo la relación de las clases de nuestro programa, los casos de uso desde el punto de vista de un usuario o el diagrama de flujo que sigue el programa.

4.8 LaTex

Sistema de composición de textos que permite la creación de documentos en formato estructurado y con eso se facilita el uso de elementos matemáticos y pseudocódigo en la memoria.

5 Metodología

Durante todo este proyecto vamos a explicar el desarrollo del código que habilita la funcionalidad de [1] y para ello antes de nada vamos a explicar la metodología que se seguirá en todo el desarrollo.

5.1 Agilidad

Utilizaremos los que se conoce como metodología ágil para realizar este proyecto, la metodología ágil se lleva usando desde el principio del desarrollo software, pero no fue hasta el 2001 que se formalizó a través del Manifiesto Ágil, el cual da una serie de pautas o consejos para el desarrollo de software ágil. La metodología ágil se basa en implementar un proceso iterativo e incremental en el desarrollo de la funcionalidad del proyecto, a diferencia del modelo en cascada que es más rígido y no es iterable.
Grafica agilidad
Grafica cascada
El enfoque ágil es recomendable en el desarrollo de proyectos software, ya que a diferencia de proyectos físicos en los proyectos software los cambios de diseño tienen un coste muy limitado. Esto facilita las modificaciones a lo largo del desarrollo que es lo que propone la metodología ágil.
En nuestro caso el uso de esta metodología no va a tener tanto impacto como podría ya que de desarrollo del proyecto lo va a realizar una sola persona. Esto quiere decir que la implementación de técnicas de organización del proyecto tendrán impacto limitado, aun así la implementación de estas practicas merecerá la pena a largo plazo.

5.2 Tablero Kanban

Posiblemente la practica más significativa en el desarrollo de un proyecto ágil consiste en la creación de un Tablero Kanban. Este tablero es una herramienta que permite visualizar el fujo de trabajo del proyecto.
tablero kanban generico
La composicion como podemos ver en la imagen anterior consiste en dividir el tablero en columnas, cada una de estas columnas contienen las tareas repartidas dependiendo de su estado actual.

1. La columna TO DO contiene las tareas que ya han sido formuladas

como requisitos que tiene que cumplir el código pero todavía no se ha empezado a trabajar en ellas.

2. La columna IN PROGRESS contiene las tareas cuya funcionalidad se esta incluyendo en el proyecto en estos momentos.

3. La columna DONE contiene las tareas que ya han sido implementadas en el proyecto.

El Tablero Kanban, si le usamos en un entorno ágil, permite ser customizado como mejor convenga al equipo de desarrollo. Por ejemplo, si estamos hablando de un proyecto grande puede que convenga añadir otra columna entre IN PROGRESS y DONE que podemos llamar por ejemplo VALIDATION que pueda contener las tareas que ya están terminadas pero requieren confirmación de otro miembro del equipo para que se cierren. **tablero kanban validation**
A continuación mostraremos como se ha utilizado y modificado el concepto de Tablero Kanban en este proyecto.
Haremos la creación y seguimiento del tablero mediante Trello, un software de gestión de proyectos que mediante su interfaz de web permite fácilmente crear y customizar Tableros estilo Kanban. A continuación se muestra el tablero en un estado avanzado del desarrollo.
tablero kanban real
Esta es una visión global del tablero por lo que es difícil ver todo lo que esta pasando, así que vamos a explicarlo parte por parte.

5.2.1 Nucleo

tablero kanban columnas principales
Esta es la parte principal del tablero y como podemos ver se parece mucho al esquema general, tenemos una columna TO DO con las tareas que no están empezadas, una columna DOING con las tareas en desarrollo.
El cambio más notable de este tablero es la adición de una columna PERMANENT que aquí se utiliza para tener siempre en mente ciertas tareas, como puede ser "Implementar tests" para intentar maximizar la cobertura de los tests, o "Tomarse la memoria en serio" para recordar tomar notas durante el desarrollo del proyecto que ayude a la creación de la memoria.
Por último tenemos dos tablas que tienen la misma razón de ser, tanto la columna MEMORIA como la columna DONE(18/4/22–24/4/22) se utilizan para almacenar las tareas que ya se han terminado, la diferencia es que en la columna DONE se almacenan las tareas que añaden valor al proyecto, y

en la columna MEMORIA se guardan las tareas relacionadas con la creación de la memoria.

5.2.2 Tareas

Con las tareas nos referimos a cada uno de los items que se mueven a lo largo de las columnas de la tabla. Y por lo general lo que intentamos hacer con estas tareas es que cada una sea unitaria, es decir, que cada tarea pueda cumplir su funcionalidad sin necesidad de que las demas tareas cumplan la suya. Esto ademas nos permite testear la funcionalidad de cada tarea por separado sin depender de otras tareas.
El problema es que estas tareas con los requisitos anteriores se nos pueden complicar mucho, o aumentar mucho de tamaño como podemos ver con el siguiente ejemplo de la tarea "Conectar App con la funcionalidad"
tablero kanban tarea
Esta tarea engloba todo el proceso de la salida de los datos por pantalla, y como hay dos formas de mostrar los datos en nuestra aplicación utilizamos un checklist para dividir esta tarea en subtareas, y cada unos de estas subtareas se encarga de conectar una parte distinta de la aplicación. Esto lo hacemos de este forma ya que Trello, a diferencia de otras herramientas como Jira, no permite la relación entre tareas, es decir, si creo as subtareas como items por si mismas no tengo forma de agruparlas bajo la misma funcionalidad.
Además de esto, dentro de la tarea tenemos una pequeña descripción de la funcionalidad que se quiere implementar y tenemos asignadas a la tarea una serie de etiquetas, en este caso son "Prioridad Alta" y "Programar" que implican que la tarea conlleva tocar codigo, y que cuanto antes se termine mejor. Pero no solo existen esas etiquetas, a continuación mostramos todas las etiquetas que se han usado a lo largo del proyecto.
tablero etiquetas
Las primeras etiquetas muestran la importancia de la tarea determinando grados de prioridad mientras que el resto de etiquetas muestran el dominio de la tarea.

5.2.3 DONE

tablero done
Aquí mostramos el resto del tablero, y como se puede ver consiste en todas las tareas que se han terminado repartidas dependiendo del intervalo de tiempo durante el cual se han completado. Esto, junto a un fichero Exel donde se ha apuntado el tiempo dedicado por día nos permite tener un buen esquema cronológico del desarrollo del proyecto.

Aquí mostramos un segmento del fichero Exel con estos datos, pero el fichero en su totalidad estará en los apéndices.
ejemplo exel

5.3 Control de versiones

El concepto control de versiones se refiere a la gestión y almacenamiento de los distintos cambios que se hacen sobre un proyecto. Esto en la practica produce en registro de todas las versiones de un proyecto a lo largo del tiempo. Esto puede hacerse de varias maneras, la forma más rudimentaria que todo el mundo ha utilizado alguna vez es hacer el proceso de forma manual. Lo que consiste en duplicar los fichero del proyecto y guardarlos aparte, ya sea un USB, un disco externo, o el propio ordenador de trabajo. Esto se suele hacer como copia de seguridad del proyecto por si ocurre algo irreparable en el proyecto real, pero en cuando requerimos de mas instancias del proyecto guardadas nos topamos con problemas de organización y espacio muy rápidamente.
ejemplo copias manuales
Por esta razón es recomendable utilizar una herramienta de control de versiones que facilite la gestión de las instancias. Existen una gran cantidad de opciones pero nosotros hablaremos de una de las más populares, la cual es la que utilizaremos nosotros en este proyecto,esa herramienta es Git.
Git permite alojar nuestro proyecto en un repositorio ajeno a nuestro ordenador lo que implica que el proyecto permanecerá seguro pase lo que pase con el ordenador desde el que se trabaja. El desarrollo del proyecto ocurre sobre una imagen local del proyecto que hay en el repositorio, y todo cambio que se quiera hacer permanente se tiene que subir al repositorio, a continuación mostramos como se hace ese proceso.
ejemplo git comandos
Tan solo con proporcionarnos esta funcionalidad Git prueba que merece la pena ser utilizado, pero esto no es todo lo que puede hacer Git, ya que, bien utilizado Git permite la coordinación de equipos de desarrollo enteros sobre el mismo repositorio gracias a múltiples de sus herramientas, pero como este proyecto se ha desarrollado en solitario no nos meteremos en este tema. Es más, en el desarrollo de este proyecto ni siquiera vamos a utilizar Git por linea de comando ya que podemos instalar un plugin sobre Spyder que nos permite tener una pequeña interfaz mediante la cual gestionar la conexión con el repositorio Git como podemos ver a continuación.
intefaz git Spyder
Esta interfaz nos permite hacer los commits al repositorio de forma mucho

más facil, y hasta podemos ver el registro de cambios. Y todo lo que hagamos desde esta intefaz se ve reflejado en el repositorio. A continuación mostraremos el repositorio visto desde GitHub web para ver la similitud de las dos interfaces.

intefaz github

Se puede ver que GitHub web tiene una interfaz más refinada, pero la información del repositorio es la misma en ambas interfaces, y desde Spyder es mucho más fácil realizar los commits, por lo cual como no queremos instalarnos otra aplicación para realizar los commits como puede ser GitHub Desktop o Fork realizaremos los commits desde Spyder.

Hablar de como volver a versiones antiguas y comandos a utilizar

Antes de terminar de hablar del control de versiones merece la pena mencionar que la implementación de Git en este proyecto se ha hecho bastante tarde ya que en un principio no se veía la utilidad, pero después de dos dolorosas sesiones de soluciones de errores que se alargaron más de la cuenta al romper partes del código que funcionaban como debían en los intentos de solucionar los errores se decidió añadir Git. En retrospectiva se debería haber usado Git desde el principio, pero por lo menos esto se tendrá en cuenta en proyectos futuros.

5.4 Tests

Antes de nada vamos a explicar a que nos referimos con tests y porque merece la pena implementarlos. Se conoce como test a un proceso automático que asegura el correcto comportamiento de una sección de código. Esto tiene una gran cantidad de ventajas muy importantes, como puede ser el proveer una detección de errores más rápida que la manual o el aislamiento del error al trozo de código que lo produce. Por lo general tener una baraja solida de tests es indispensable para el optimo desarrollo de un proyecto software, pero no nos podemos olvidar de que los tests consisten de más código que hay que mantener, así que es indispensable mantener los tests simples para limitar la carga técnica que se añade al proyecto. En la mayoría de los casos es mejor tener una mayor cantidad de test, los cuales individualmente son más simples, que tener muy pocos test muy complejos. En el desarrollo de este proyecto vamos a tener muy en cuenta los test para facilitar el desarrollo y sostenibilidad del proyecto, por eso vamos a implementar una practica de programación llamada TDD o Test-Driven Development (Desarrollo Dirigido por Tests) que nos ayudara a asegurar que los test cubran toda la funcionalidad del proyecto, nunca se podrá llegar a una cobertura del 100%, pero esta practica ayuda a acercarnos a esa meta.

Gráfica TDD
Explicar TDD
Hay una alta variedad de categorías de test como pueden ser:
Categorías de tests
Pero nosotros solo vamos a utilizar los tests unitarios ya que nuestro proyecto es lo suficientemente simple como para obviar los demás tipos de tests, siempre se pude en un futuro añadir otras clases de tests.
Los tests los implementaremos ayudándonos de otro plugin de Spyder llamado Spyder-unittest que provee las herramientas necesarias para una fácil creación de tests.
Ejemplo he imágenes del plugin de tests de Spyder
El uso de TDD se ha mantenido a lo largo de la mayor parte del desarrollo del proyecto, produciendo tests para la gran mayoría de clases funcionales del código, pero con la introducción de una interfaz en nuestro proyecto nos fue imposible mantener las pautas dadas por TDD como mostraremos a continuación.

5.5 Interfaz

Se denomina interfaz a la conexión funcional entre dos sistemas o componentes que crea una vía para facilitar el intercambio de información. Para nosotros la conexión se efectúa entre el usuario y el programa mediante una interfaz gráfica muy simple como podemos ver a continuación.
Imagen interfaz gráfica
Esto permite al usuario interactuar de forma intuitiva con el programa.
Para desarrollar esta interfaz hemos buscado alternativas simples para crear aplicaciones pyton, y de esto forma a lo primero que llegamos fue a la librería gráfica **tkinter** ya que en su momento parecía la opción más simple, pero a lo largo del desarrollo de la aplicación se empezaron a ver sus limitaciones,la más relevante entre ellas es que tkinter no permite trabajar con htmls que es el formato en el cual generamos las gráficas con los resultados, por lo que en este punto hay que hacerse la pregunta,¿ modificamos el formato de las gráficas para poder seguir trabajando con tkinter o buscamos otra alternativa para hacer la interfaz gráfica?. Lo más fácil habría sido modificar el formato de las gráficas, pero en mi opinión esto es una mala practica ya que estamos modificando una parte del código que funciona perfectamente para acomodarnos a las restricciones de la interfaz, por lo que vamos a intentar buscar otra alternativa para crear la interfaz gráfica.
La alternativa que encontramos es utilizar la librería gráfica **PyQt5** que además tiene la ventaja de ser compatible con la aplicación Qt Designer,

un entorno de desarrollo gráfico permite crear y modificar la interfaz visualmente.

Imagen Qt Designer

Tutorial Qt Designer

Con PyQt5 como alternativa a tkinter ya podemos migrar la interfaz a PyQt5, pero en este punto se hace aparente un fallo de diseño que hemos tenido hasta ahora. La conexión de la interfaz gráfica con la funcionalidad esta implementada de la siguiente forma:

Diagrama interfaz conexion todas clases

Este diagrama muestra como la propia interfaz se comunica ella misma con cada una de las clases del procesos. Esto es un gran problema ya que ahora para migrar de interfaz hay que cambiar todas las conexiones haciendo que todo el código implementado para tkinter se tenga que desechar. Esto se podría haber solventado implementando una arquitectura modelo vista controlador, haciendo que la conexión entre la interfaz y la funcionalidad tenga esta pinta:

Diagrama interfaz modelo vista controllador

Esto no se implementó antes debido al limitado tamaño del proyecto, pero en retrospectiva habría sido lo más adecuado, no solo por la migración de la interfaz sino también por los tests.

Cuando empezamos a desarrollar la interfaz nos topamos con el problema de que la librería de tests unitarios que utilizamos no estaba capacitada para tratar con la creación de aplicaciones, habríamos necesitado implementar un entorno de tests de integración, lo que nos resultaba demasiado complejo en ese punto. Por eso mismo violamos las pautas del TDD en este punto, pero ahora gracias a la decisión de diseño modelo vista controlador podemos testear la conexión entre la interfaz y la funcionalidad simplemente creando tests unitarios sobre los controladores. Por ejemplo, queremos testear que la selección de empresas en el frontend (Interfaz) queda reflejada en el backend, para esto solo necesitamos hacer un test unitario sobre el InputController.

Imagen explicativa del ejemplo

Así que podemos decir que gracias a implementar un diseño modelo vista controlador pudimos volver a cumplir las pautas del TDD.

6 Arquitectura

6.1 Fundamentos

En un proyecto software consideramos que la arquitectura es el diseño de más alto nivel de la estructura del proyecto en cuestión. Para la creación de esta arquitectura se utilizan patrones de diseño y técnicas de abstracción que permiten al proyecto software realizar su función de la forma más optima. Con la función del sofware no nos referimos literalmente a que el software pueda realizar su función principal, eso es lo esperable con o sin una arquitectura bien definida, nos referimos a las funcionalidades secundarias del código como puede ser:

1. Sostenibilidad del código: Con esto nos referimos a la dificultad que reside en mantener el código del proyecto. Si el código es poco sostenible el coste, tanto económico como de complejidad, aumenta a lo lago del tiempo de forma descontrolada. Si el código es sostenible este aumento se pude reducir lo que en la practica extiende la vida útil del software. **Grafica software sostenible vs no sostenible**

2. Facilidad de expansión: Esto tiene que ver con el coste de implementar nueva funcionalidad a un código ya en producción.. A mayor facilidad de expansión menos aumenta el coste de cada sucesiva ampliación de funcionalidad, esto está ligado a la arquitectura del proyecto.

3. Seguridad: La seguridad de un software se refiere a la resistencia de dicho software a ataques. Estos ataques pueden ser de cualquier tipo, pueden ser intentos de acceso a información privada o ataques DDos(denegación de servicio) por mencionar algunos. Para prevenir estos ataques es necesario tener implementada una arquitectura segura.

4. Optimización de recursos: Este punto es el más fácil de explicar pero en la practica es sin duda uno de los más complejos. Simplemente consiste en minimizar los recursos, tanto de tiempo como de almacenamiento de datos, que se utilizar en la funcionalidad del proyecto.

Dependiendo del tipo de proyecto en el que estemos se priorizaran de distinta forma estas metas. Por ejemplo si consideramos una aplicación de banca se prioriza la seguridad, pudiendo dejar en un segundo plano la optimización de recursos y eso se ve reflejado en la arquitectura del proyecto.
En nuestro caso priorizaremos la sostenibilidad del código y la facilidad de expansión sin tener muy en cuenta la seguridad ya que no trabajamos con

ninguna clase de información delicada.

6.2 Modelo Vista Controlador

Ahora vamos a hablar de la mayor decisión arquitectónica que hemos implementado en este proyecto la arquitectura Modelo Vista Controladore (MVC), primero hablaremos en que consiste desde un punto de vista puramente teórico, y luego mostraremos como lo hemos implementado en el proyecto.
La arquitectura MVC se centra en controlar el flujo de datos dentro del proyecto, define tres componentes distintos, Modelo, Vista y Controlador. Cada componente se ocupa de una tarea distinta y las conexiones entre los tres componentes están altamente restringidas.

1. El **Modelo** es lo que consideramos el backend del proyecto, es la parte del código que se encarga de toda la lógica del proceso, es decir, el acceso a los datos almacenados y el tratamiento de esos datos. Se puede considerar el núcleo del proceso y como tal suele tener la mayor parte de la complejidad computacional del proceso.

2. La **Vista** se refiere a la interfaz de usuario y se encarga tanto de proveer un medio para mostrar información al usuario, como una ventana mediante la cual el usuario pueda interactuar con el proceso, lo podemos considerar el frontend de la aplicación.

3. El **Controlador** es el intermediario entre el Modelo y la Vista, provee una vía para transportar los datos, y si es necesario transforma los datos para que el Modelo y la Vista se puedan entender. Es la parte del proceso más flexible, puede simplemente ser el link entra las otras dos capas, o puede ser el cerebro de la operación si se le añade un mínimo de lógica para que pueda controlar el flujo de la información.

Esquema modelo vista controlador En nuestro proyecto tendremos la siguiente separación, mostrada intuitivamente en la organización de las carpetas del proyecto.
Imagen organización carpeta src Tenemos una carpeta con los controladores separados según la parte del proceso que gestionan, el InputController se encarga de recoger la información de entrada decidida por el usuario, el ExecutionController se encarga de llamar a la ejecución del programa, y tanto PlotController como ResultController se encargan de mostrar los resultados al usuario, por ultimo tenemos AltairPlot separado del PlotController por si en un futuro se quiere cambiar el tipo de plot que se muestra por pantalla

de Altair a otro formato.
Por otro lado tenemos la carpeta que contiene el modelo, en la cual nos meteremos más adelante, y por ultimo tenemos la vista, que tiene tanto la interfaz PyQt5 en QtApp_ ui.py con su esquema en QtApp.ui como la antigua interfaz tkinter en App_ Old.py.
Esto nos genera un proyecto con la siguiente arquitectura.
Diagrama modelo vista controlador proyecto

6.3 Estructura de Modelo

Dentro del modelo tenemos la siguiente estructura de carpetas:
imagen fichero modelo No vamos a meternos todavía en la funcionalidad especifica de cada clase, lo que si que vamos a hacer es representar las conexiones mediante el siguiente diagrama de clases.
Diagrama de clases Explicacion diagrama de clases

6.4 Estructura completa

Con el núcleo del programa ya explicado podemos dar un paso atrás y ver la estructura global del proyecto.
Imagen fichero total Ahora explicaremos cada uno de los elemento que tenemos.

1. **config**: Aquí se guardan las variables universales de todo el código, como son el día actual o la ruta del proyecto.

2. **documentation**: Aquí se guardan todos los ficheros informativos del proyecto. Por un lado tenemos los fichero utilizados para hacer las memorias del proyecto,los ficheros TEX que generan las memorias en PDF,los recursos utilizados en estas memorias y los propios PDFs. Y por el otro lado tenemos un par de ficheros donde se han ido tomando apuntes a lo largo del desarrollo del proyecto como son Bibliografia.txt y TFG_ Horario.ods.

3. **resources**: Esta carpeta contiene los recursos que utiliza o genera el proyecto. En data están los ficheros csv que hacen el papel de base de datos del proyecto, en graphs tenemos todos los gráficos que hemos generado en el desarrollo de este proyecto, output contiene los resultados que hemos utilizado para el estudio presente en [1] y App.ico es el icono de la interfaz gráfica.

4. **src**: Aquí esta el código del proyecto que ya hemos mencionado antes.

5. **tests**: Aquí tenemos todos los tests que se han ido creando a lo largo del desarrollo, todos estos tests son unitarios y cada fichero se encarga de testear la clase que indica su nombre.

6. **Otros ficheros**: Por ultimo tenemos el fichero ejecutable QtApp.py que nos abre la interfaz gráfica de la aplicación y el README.md que contiene un resumen del proyecto.

En conjunto esto nos crea la siguiente estructura de proyecto:
Diagrama modelo vista controlador proyecto con tests La información más importante que recordar de este diagrama es que los tests y el modelo acceden a datos distintos, esto es un punto muy importante ya que permite el total desacoplamiento de los tests. **Explicacion diagrama de componentes**

7 Desarrollo

Con la estructura explicada podemos empezar a hablar del desarrollo en sí. Vamos a dividir el desarrollo en 3 etapas las cuales son fácilmente diferenciables debido a notables mejoras en la gestión del trabajo a lo largo del tiempo.

1. **Fase 1**: Etapa inicial en la que para documentar y gestionar el proyecto tan solo se utilizó un documento Exel para tomar notas.
2. **Fase 2**: Etapa intermedia en la que se mejoró la gestión de las tareas añadiendo un tablero Trello.
3. **Fase 3**: Ultima etapa en la que por fin se añade un sistema real de control de versiones como es Git.

7.1 Fase 1 (Exel)

Aquí tenemos documentado todo lo que se hizo en esta fase:
Foto Exel Engloba un periodo de 3 meses en los cuales se realizan unas 84 horas de trabajo distribuidas de la siguiente forma:

1. **Investigación**: 21 horas investigando recursos y conceptos
2. **Programación**: 45 horas escribiendo codigo
3. **Organización**: 15 horas planteando o cambiando la estructura del proyecto
4. **Memoria**: 3 horas escribiendo documentación
5. **Fix**: 0 horas arreglando buggs del código

Grafica de tiempo Fase 1
La mayor parte del tiempo es utilizado programando, en este punto se crean las primeras versiones de los objetos que utilizaremos en el algoritmo.Estos objetos son SimpleWave y ComplexWave que son los primeros objetos que contienen la serie temporal que se le pasa al algoritmo como input. También tenemos las transformaciones que se aplicaran a las series temporales como son Fourier, Wavelet y VisibilityeGraph. Y se importa el algoritmo K-means sin modificar en la clase K-mean que servirá de base para nuestro algoritmo. Por ultimo se crea una vía para recoger datos de stocks que se convertirán

en series temporales.

La estructura del proyecto es la siguiente al final de esta fase:,
Diagramas de clases Fase 1 Lo más relevante es que en este punto los tests están acoplados al resto del código, esto se debe principalmente a que aunque hemos estado siguiendo un desarrollo TDD, no hemos utilizado herramientas que facilitan la creación y uso de los tests como puede ser el plugin unittest de Spyder que utilizaremos en las siguientes fases. Esto quiere decir que los tests han sido más costosos de crear, y lo que es todavía más relevante, los tests han tenido que ser ejecutados a mano uno a uno hasta ahora.
Por otro lado podemos ver que las clases todavía no estas conectadas entre ellas.

7.2 Fase 2 (Trello)

Aquí ya tenemos implementado el tablero Trello para gestionar las tareas a realizar:
Foto Trello Engloba un periodo de 3 meses en los cuales se realizan unas 272,5 horas de trabajo distribuidas de la siguiente forma:

1. **Investigación**: 39 horas investigando recursos y conceptos
2. **Programación**: 155,5 horas escribiendo código
3. **Organización**: 29 horas planteando o cambiando la estructura del proyecto
4. **Memoria**: 18 horas escribiendo documentación
5. **Fix**: 31 horas arreglando buggs del código

Grafica de tiempo Fase 1
En esta fase seguimos manteniendo que la mayor parte del tiempo se ha usado programando, lo cual no es extraño ya que en esta fase completamos la funcionalidad del algoritmo en su totalidad.
Lo primero que hacemos es implementar herencia sobre ComplexWave, Graph y K-mean lo que nos permite generalizar la funcionalidad de estos objetos.
Diagrama herencia
Luego desarrollamos pequeñas customizaciones sobre K-means sirviéndonos de que acabamos de implementar herencia, y conectamos las clases para tener la primera versión funcional del algoritmo.

Diagrama kmeans y conexion
Creamos la estructura de pruebas que en un futuro nos servirá para realizar las simulaciones del estudio **Diagrama resultados estudio**
Y por ultimo, la parte más importante de esta fase, modificamos la estructura del K-means utilizado buscando que nuestro algoritmo tenga resultados más consistentes que el K-means inicial.
Explicar kmeansciclico kmeans graph
Diagrama kmeansciclico kmeans graph

7.3 Fase 3 (Git)

Aquí ya tenemos implementado un sistema de control de versiones Git ya que en la fase anterior dedicamos 31 horas a solucionar buggs, y eso lo podríamos haber reducido considerablemente con control de versiones:
Repositorio git Engloba un periodo de 3 meses en los cuales se realizan unas 178,5+ horas de trabajo distribuidas de la siguiente forma:

1. **Investigación**: 26,5 horas investigando recursos y conceptos
2. **Programación**: 52 horas escribiendo código
3. **Organización**: 15 horas planteando o cambiando la estructura del proyecto
4. **Memoria**: 100+ horas escribiendo documentación
5. **Fix**: 3 horas arreglando buggs del código

Grafica de tiempo Fase 1
En esta fase tenemos menos horas dedicadas a programación en comparación a las fases anteriores ya que el algoritmo se termino de crear en la segunda fase. En esta fase lo único que hacemos con respecto a escribir código es crear una interfaz de usuario simple y conectarla con el resto del código creando controladores para seguir una arquitectura modelo vista controlador.
Interfaz
Diagrama de clases
Por otro lado a lo que más tiempo dedicamos en esta fase es a escribir las memorias del proyecto.
Cabe resaltar que el tiempo dedicado a resolver buggs ha disminuido considerablemente, en parte por dedicar menos tiempo a programar,pero sin duda el control de versiones ha ayudado a mantener este numero bajo.

7.4 Casos de Uso

Ahora mostraremos algunos casos de uso a través de la interfaz para mostrar lo que puede hacer nuestro proyecto.
Primero ejecutamos la funcionalidad estándar.
Diagrama funcion estandar
Resutados en interfaz
Ahora modifiquemos el intervalo de tiempo y ejecutemos la funcionalidad rápida(la funcionalidad estándar con el input transformado con Wavelet).
Diagrama kmeansciclico kmeans graph
Diagrama funcion rapida modificando el tiempo
Resutados en interfaz

Ahora modifiquemos las empresas de entrada y ejecutemos la funcionalidad completa(la funcionalidad estándar con el input transformado a Grafo de Visibilidad).
Diagrama funcion completa modificando las empresas
Resutados en interfaz

8 Conclusiones

9 Bibliografía

[1] Javier Méndez García-Brioles (2022), *Tituo*

[2] Gao, X., Xiao, B., Tao, D. et al. A survey of graph edit distance. Pattern Anal Applic 13, 113–129 (2010). https://doi.org/10.1007/s10044-008-0141-y

[3] Pinar Sanz Sacristán(2018), *ESTUDIO DE ALGORITMOS DE COMPUTACIÓN BASADA EN GRAFOS APLICADO AL ANÁLISIS DE REDES* , https://repositorio.uam.es/bitstream/handle/10486/688032/sanz_sacrist%C3%A1n_pinar_tfg.pdf?sequence=1&isAllowed=y

[4] Ricardo Rosenfeld y Jerónimo Irazábal (2013), *Computabilidad, complejidad computacional y verificación de programas*, http://190.57.147.202:90/jspui/bitstream/123456789/988/1/computabilidad-complejidad-verificacion.pdf

[5] Magdalena Ferrán Aranaz (2011), *UNA METODOLOGÍA DE MINERÍA DE DATOS PARA LA AGRUPACIÓN DE SERIES TEMPORALES: APLICACIÓN AL SECTOR DE LA CONSTRUCCIÓN RESIDENCIAL.* , https://eprints.ucm.es/id/eprint/12804/1/T32968.pdf

[6] Regino Criado, Miguel Romance y Luis E. Solá (2014), *Teoría de Perron-Frobenius: importancia, poder y centralidad*, https://gaceta.rsme.es/abrir.php?id=1217

[7] Keshi Liu, Tongfeng Weng, Changgui Gu, Huijie Yang, Visibility graph analysis of Bitcoin price series, Physica A: Statistical Mechanics and its Applications, Volume 538, 2020, 122952, ISSN 0378-4371, https://doi.org/10.1016/j.physa.2019.122952

[8] Lacasa, Lucas & Luque, Bartolo & Ballesteros, Fernando & Luque, Jordi & Nuño, Juan Carlos. (2008). From time series to complex networks: The visibility graph. Proceedings of the National Academy of Sciences. 105. 4972. 10.1073/pnas.0709247105.

[9] Agbinya, Johnson. (1996). Discrete wavelet transform techniques in speech processing. 514 - 519 vol.2. 10.1109/TENCON.1996.608394.

[10] D. Song, A. M. Chung Baek and N. Kim, "Forecasting Stock Market Indices Using Padding-Based Fourier Transform Denoising and Time Series Deep Learning Models," in IEEE Access, vol. 9, pp. 83786-83796, 2021, doi: 10.1109/ACCESS.2021.3086537.

[11] S. M. Idrees, M. A. Alam and P. Agarwal, "A Prediction Approach for Stock Market Volatility Based on Time Series Data," in IEEE Access, vol. 7, pp. 17287-17298, 2019, doi: 10.1109/ACCESS.2019.2895252.

[12] CORREDOR, PILAR, SANTAMARÍA, RAFAEL PREDICCIÓN DE VOLATILIDAD Y PRECIOS DE LAS OPCIONES EN EL IBEX-35. Revista de Economía Aplicada [en linea]. 2001, IX(25), 39-64[fecha de Consulta 8 de Junio de 2022]. ISSN: 1133-455X. Disponible en: https://www.redalyc.org/articulo.oa?id=96917680002

[13] Regino Criado Herrero, Roberto Muñoz Izquierdo (2007), *Un semestre de matemática discreta*

https://www.econstor.eu/bitstream/10419/238381/1/756.pdf

10 Apéndices