ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN xxx

Curso Académico 20xx/20xx

Trabajo Fin de Grado

ESTUDIO DE DIVERSIFICACIÓN DE INVERSIONES SOBRE IBEX-35 UTILIZANDO CLUSTERING

Autor:
Javier Méndez García-Brioles
Directores:
Regino Criado

Indice

Contents

Resumen

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.

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 pseudo-distancias, 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.

Fundamentos Teóriocos

Ondas Temporales

Ondas Simples

LLamaremos ondas simples a las culaes compuestas exclusivamente por una funcion seno, con sus posibles modificaciónes de amplitud, frecuencia, y desplazamiento.

Ondas Compustas

LLamaremos ondas compuestas a la superposición de ondas simples a lo largo del tiempo

Transformada de Fourier

La transformada de Fourier es una función matematica que permite relacionar un dominio temporal con un dominio de frecuencias en ambos sentidos

Transformada Wavelete

La transformada Wavelete es una función muy utilizada en compresión de imagenes que permite dividir la cantidad de valores manteniendo la mayor cantidad posible de detalle.

Teoria de Grafos

Grafo

Un grafo ($G = (V, E)$) es una estructura de datos compuesta por vertices (V), donde se almacenan los datos, y aristas o arcos ($E = V \times V$), que representan las conexiones entre dos vertices.

Grafo no Dirigido

Grafo ($G = (V, E)$) tal que:

$$V \neq \emptyset$$

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

Grafo Dirigido
Grafo ($c = 1$) tal que:
Vertice
Componente minimo indivisible por el que estan compuestos los grafos
Arista
Componente de un grafo que se caracteriza por conectar dos vertices (E)
Arco
Equivalente a la arista de un grafo no dirigido en un grafo dirigido
Grado
El grado de un vertice ($v \in V$) 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 segun que nodos se ven entre ellos sin que otro nodo les bloque la visión. Esta restrincción equivale a cumplir la equación ($y_m + \frac{y_n - y_m}{n-m}(k-m) \geq y_k$) para todo nodo k entre m y n ($m < k < n$)
Algoritmos sobre grafos
Algoritmo de Dijkstra
Tambien conocido como algoritmo de minimos caminos nos permite determianar 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 minima distancia entre todos los pares de nodos. **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 traves 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 puden usar para determinar la centralidad, pero nosotros nos vamos a centrar en la centralidad de grado y la interpolación

Centralidad de grado

La centralidad de grafo es posiblemente la medida de centralidad más facil 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 utiles para el estudio de grafos, pero para ciertas aplicaciónes 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 ocasionaria a la red. Por lo tanto, con esta información podriamos 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.

K-means

El K-means o K-medias es un método de clasificación no supervisada que pemite 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.

K-means consta de 3 partes indispensables:

1. **Inicialización de centrodies**: 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. Pero hay que tener en cuenta que este proceso esta pensado para trabajar con distancias, así que si se utilizan pseudodidstancias hay que tomar medidas extras en este paso.

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 culster del centroide correspondiente.

3. **Actualizar centrodies**: 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 condicion 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 pseudocodigo:
La principal ventaja del k-means es que es muy rapido, pero tiene tambien

Algorithm 1 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;

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.

Distancias

Una distancia es cualquier funcion que comple 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 analoga por la desigualdad triangular:
$d(x, y) \geq 0$ (no negatividad)

Pseudodistancias

Este termino se le da a las distancias que no cumplen la restricción 1

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

que se reduce a

$$d(x, x) = 0$$

por lo tanto es posible tener dos puntos distintos que tengan distancia 0.

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 estandar.
Definimos volatilidad como:

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

****

Probabilidad

No vamos a necesitar mucho del ambito de la probabilidad pero aun así necesitaremos conocer añgunos conceptos basicos:
-(Conceptos necesarios hasta llegar al Teorema de Bayes)

Complejidad Computacional

???

****

Herramientas

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

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.

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.

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 Analisis: 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 ejecucion de la aplicación. El tiempo de ejecución se separa por método, lo que permite ver que metodos estan consumiento más

recursos y esto ayuda en el proceso de optimización del código.

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 libreria grafica QT en python lo que nos da opciones útiles que no nos dan otras librerias de interfaces de python como puede ser tkinter.

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.

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.

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.

LaTex

Aplicación que permite la creación de memorias facilitando el uso de elementos matemáticos y pseudocódigo.

1 Proceso

(incluyendo descripción del problema, estudio de alternativas y metodología empleada)

El unico problema es que el algoritmo K-means no esta preparado para trabajar con pseudodistancias por lo que hay que hacerle unas modificaciones para evitar problemas.

1.0.1 Modificaciones

2 Descripción informática

(puede incluir especificación, diseño, implementación y pruebas).

3 Experimentos / validación

4 Conclusiones

(incluyendo los logros principales alcanzados y posibles trabajos futuros)

5 Bibliografía

6 Apéndices