UNIVERSIDAD DEL ROSARIO

TESIS DE MAESTRÍA

Administración Algorítmica de Portafolio con Hidden Markov Models

Autor:
Juan Felipe ROMERO RAMIREZ

Director de Tesis:
Hugo Eduardo RAMIREZ JAIME

Tesis presentada para aspirar al grado en la Maestría en Finanzas Cuantitativas

en la

Universidad del Rosario
Facultad de Economía

Junio de 2019

"Finalmente pude llegar a escribir esta sección... Este sería un agradecimiento, pero realmente es más que eso. Esto esto está dedicado a mis abuelos, mi mamá y mi hermano. Sin lugar a duda este es un logro que quiero compartir con aquellas personas que me han acompañado y apoyado para llegar acá: Danna, Ferno, Ralphie, mis amigos de la universidad y colegio, gracias por estar ahí y ser parte de esto."

Juan Felipe Romero Ramirez

UNIVERSIDAD DEL ROSARIO

Resúmen

Facultad de Economía

Maestría en Finanzas Cuantitativas

Administración Algorítmica de Portafolio con Hidden Markov Models

por Juan Felipe ROMERO RAMIREZ

Esta tesis implementa una estrategia de trading de alta frecuencia, conocida como *pairs trading*, sobre los activos de un portafolio haciendo uso de diversos algoritmos de *machine learning*. Se hace énfasis en el uso de los *Hidden Markov Models* para mejorar la estrategia de trading y la administración del portafolio.

Keywords: Pairs trading, Machine Learning, Hidden Markov Models, Black-Litterman.

Tabla de Contenidos

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Gracias al alto volumen de datos, disponibilidad de éstos, y avances en los campos de la informática, matemáticas y estadística, se han desarrollado modelos (algoritmos) de aprendizaje para resolver problemas de alta complejidad. Estos modelos requieren ser entrenados para mejorar su desempeño por medio de una muestra significativa de los datos que se tienen disponibles. Los modelos de aprendizaje se clasifican a partir del grado de intervención previa. Por lo tanto, si se conocen previamente los criterios y posibles outputs, el modelo es supervisado, por otra parte, si el algoritmo tiene criterio propio para modelar es considerado no supervisado.

Los Hidden Markov Models o Modelos de Estados Escondidos de Markov (HMMs) han sido usados para identificar y estimar regímenes o estados ocultos dentro de ciertos procesos estocásticos. Este modelo es considerado como no supervisado gracias a sus características, metodología y outputs.

Los HMM han sido usados aproximadamente durante tres décadas para procesos y aplicaciones de detección y procesamiento de señales. Éstos han sido utilizados en diversos campos de la ciencia, como son:

- Bioinformática: análisis de secuencias biológicas.
- Ciencias Naturales: predicción de lluvia, terremotos y dirección del viento.
- Finanzas: análisis y modelación de series de tiempo financieras, enfoque desarrollado en este trabajo.
- Biofísica: modelación de canales de iones.
- Biología: análisis del comportamiento animal.
- Medicina: reconocimiento de señales de la expresión, escritura y habla.

Los HMMs son modelos en los cuales la distribución que genera una observación depende de un estado, el cual a su vez es dependiente de un proceso de Markov subyacente no observado. Para llegar a la definición teórica y propiedades de un HMM es necesario entender los siguientes axiomas:

- La distribución marginal de un HMM es una *mixture distribution*.
- Las cadenas de Markov proveen la definición del proceso de parámetros subyacente de un HMM.

Por ello, en el siguiente capítulo se abordarán las definiciones y propiedades de las *mixture distributions* y las cadenas de Markov. Más adelante se llegará al núcleo de

este marco teórico, definir matemáticamente un HMM y entender sus propiedades.

Este trabajo hará uso del HMM para identificar los estados ocultos en una serie de procesos estocásticos, siendo específico, sobre los procesos de precios de algunos activos financieros de Estados Unidos. Al identificar estos estados ocultos y junto con una estrategia de pairs trading se podrán capturar señales óptimas para la ejecución de operaciones en el mercado sujeto a unos límites de inversión establecidos por el modelo de Black-Litterman.

Capítulo 2

Marco Teórico

2.1 Mixture Models

Los *mixture models* son diseñados para incluir la heterogeneidad en una población, la cual está compuesta por grupos no observados. Para aplicar esta clase de modelos, se asume que cada grupo tiene una distribución distinta para la variable observable.

Cuando existe un evento que es generado por una serie de distribuciones con distintas medias, existe un mecanismo aleatorio que elige que parámetros de las distribuciones están activos, este mecanismo se conoce como **proceso de parámetros**. Si el proceso de parámetros es una serie de variables aleatorias independientes se considera una *independent mixture*.

En general, una *independent mixture distribution* consiste en un número finito m de distribuciones de componentes y una *mixing distribution* que selecciona desde estos componentes. En el caso de dos componentes, la *mixture distribution* depende de dos probabilidades o funciones de densidad:

TABLA 2.1: *Mixture Distribution* para $m = 2$

Componente	1	2
Probabilidad o Función de Densidad	$p_1(x)$	$p_2(x)$

Para especificar el componente, se necesita una variable aleatoria C discreta que lleve a cabo el mixing, para este caso con $m = 2$ se denotaría como:

$$C = \begin{cases} 1 & \text{con probabilidad} \quad \delta_1 \\ 2 & \text{con probabilidad} \quad \delta_2 = 1 - \delta_1 \end{cases}$$

Ya entendiendo para el caso de dos componentes, es válido generalizar la extensión para m componentes. Denotando $\delta_1, \ldots, \delta_m$ las probabilidades asociadas a los diferentes componentes y $p_1, \ldots, p_m$ las funciones de densidad de estos. Ahora es necesario definir X como la variable aleatoria que tiene la *mixture distribution*. En el caso discreto la función de probabilidad de X esta dada por:

$$\begin{aligned} p(x) &= \sum_{i=1}^{m} \mathbb{P}(X = x | C = i)\mathbb{P}(C = i) \\ &= \sum_{i=1}^{m} \delta_i p_i(x) \end{aligned}$$

2.1.1 Momentos

La esperanza de la *mixture* puede ser denotada en terminos de la esperanza de las distribuciones de los componentes. Siendo Y_i una variable aleatoria con función de probabilidad p_i, se tiene:

$$\mathbb{E}(X) = \sum_{i=1}^{m} \mathbb{P}(C = i)\mathbb{E}(X|C = i) = \sum_{i=1}^{m} \delta_i \mathbb{E}(Y_i)$$

Es más, para una *mixture* el $k-$esimo momento es una combinación lineal de los $k-$esimos momentos de los componentes Y_i:

$$\mathbb{E}(X^k) = \sum_{i=1}^{m} \delta_i \mathbb{E}(Y_i^k), \quad k = 1, 2, \ldots$$

2.1.2 Estimación de Parámetros

La estimación de los parámetros de una *mixture distribution* se realiza frecuentemente por medio de máxima verosimilitud (ML), para el caso de m componentes la función de verosimilitud estaría dada por:

$$L(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m, \delta_1, \ldots, \delta_m | x_1, \ldots, x_n) = \prod_{j=1}^{n} \sum_{i=1}^{m} \delta_i p_i(x_j, \boldsymbol{\theta}_i) \tag{2.1}$$

Donde $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m$ son los vectores de parámetros de las distribuciones compuestas,estas son distribuciones que siguen las variables que se representan por sumas, $\delta_1, \ldots, \delta_m$ son los parámetros de mezcla de las distribuciones, sumando uno, por último $x_1, \ldots, x_n$ son las n observaciones. Para el caso en que cada una de las distribuciones compuestas sean especificadas por un parámetro, sería necesario estimar $2m - 1$ parámetros independientes, m valores de θ y $m - 1$ valores de δ.

Teniendo en cuenta lo anterior, existen casos donde la maximización de la función de verosimilitud no tiene solución analíticas. Por ello es más conveniente realizarla mediante métodos numéricos, la alternativa más utilizada es el algoritmo EM, el cual será explicado en las siguientes secciones de este trabajo .

2.2 Cadenas de Markov

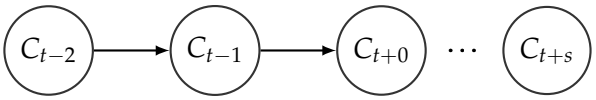
El segundo gran bloque para la construcción teórica de los HMMs son las cadenas de Markov. Para definir este concepto es necesario suponer una secuencia de variables aleatorias $\{C_t : t \in \mathbb{N}\}$, de modo que ésta secuencia es considerada una cadena de Markov (MC) si satisface la propiedad de Markov:

$$\mathbb{P}(C_{t+1}|C_1, \ldots, C_t) = \mathbb{P}(C_{t+1}|C_t).$$

Esta propiedad propone que condicionar el proceso a toda su historia es equivalente a condicionarlo a su valor más reciente, es decir a C_t, en otras palabras quiere decir que el proceso no tiene memoria. Por efectos de notación se definirá $\boldsymbol{C}^{(t)}$ como la historia $(C_1, \ldots, C_t)$, entonces la propiedad de Markov se reescribiría como:

$$\mathbb{P}(C_{t+1}|\boldsymbol{C}^{(t)}) = \mathbb{P}(C_{t+1}|C_t).$$

La propiedad de Markov puede ser considerada la primera relajación del supuesto de independencia. Las variables aleatorias $\{C_t\}$ son dependientes convenientemente para efectos matemáticos, como se puede observar en el siguiente diagrama, el pasado y el futuro son dependientes únicamente a través del presente, es decir, el proceso que cumpla la propiedad no tiene memoria más allá de un periodo.

Cadena 

Una característica principal de las cadenas de Markov son las probabilidades condicionales, conocidas como probabilidades de transición:

$$\mathbb{P}(C_{s+t}|C_s = i).$$

Estas describen la probabilidad de cambio de estado, por ejemplo, la probabilidad de que el mercado pase de un estado alcista a un estado asociado con crisis. Si estas probabilidades no dependen de s, la cadena de Markov asociada a estas es considerada **homogénea**, de otro modo no homogénea. Dado que se tenga una cadena de Markov homogénea, las probabilidades de transición son denotadas como:

$$\gamma_{ij}(t) = \mathbb{P}(C_{t+s}|C_s = i)$$

Ahora se definirá la matriz $\mathbf{\Gamma}(t)$, cuyo elemento (i, j) esta dado por $\gamma_{ij}(t)$. Esto con el fin de caracterizar todas las cadenas de Markov estado-espacio finitas homogeneas, las cuales deben satisfacer la propiedad de las ecuaciones de Chapman-Kolmogorov, tal que:

$$\mathbf{\Gamma}(t+u) = \mathbf{\Gamma}(t)\mathbf{\Gamma}(u)$$

De igual forma, las ecuaciones de Chapman-Kolmogorov implican que para todo $t \in \mathbb{N}$, se cumple:

$$\mathbf{\Gamma}(t) = \mathbf{\Gamma}(1)^t;$$

es decir, la matriz de transición de probabilidades de t es la matriz $\mathbf{\Gamma}(1)$ a la $t-$ésima potencia. La matriz $\mathbf{\Gamma}(1)$ será denotada desde ahora como $\mathbf{\Gamma}$, es una matriz de probabilidades cuadrada con sus filas sumando uno, de la forma:

$$\mathbf{\Gamma} = \begin{pmatrix} \gamma_{11} & \cdots & \gamma_{1m} \\ \vdots & \ddots & \vdots \\ \gamma_{m1} & \cdots & \gamma_{mm} \end{pmatrix},$$

donde m denota el número de estados en la cadena de Markov. La restricción que propone que las filas sumen uno puede ser escrita como $\mathbf{\Gamma 1'} = \mathbf{1'}$, es decir, que la suma de las probabilidad de transición del estado i a los otros estados y mantenerse en el estado i sea 1. La matriz $\mathbf{\Gamma}$ se conoce como la (*one-step*) matriz de transición de probabilidad (t.p.m). Este es uno de los elementos más importantes para este trabajo, ya que permitirá entender la dinámica de los estados escondidos de un HMM y tomar decisiones de inversión.

Las probabilidades incondicionales $\mathbb{P}(C_t = j)$ de una cadena de Markov estando en un estado y tiempo dado se denota por el siguiente vector:

$$\boldsymbol{u}(t) = (\mathbb{P}(C_t = 1), \ldots, \mathbb{P}(C_t = m)), \quad t \in \mathbb{N}.$$

Ahora, $\boldsymbol{u}(1)$ es la distribución inicial de la cadena de Markov. Para deducir la distribución en el tiempo $t+1$ desde t, se post multiplica por la matriz de transición de probabilidad $\boldsymbol{\Gamma}$:

$$\boldsymbol{u}(t+1) = \boldsymbol{u}(t)\boldsymbol{\Gamma}. \tag{2.2}$$

2.2.1 Distribuciones Estacionarias

Una cadena de Markov con matriz de transición de probabilidad $\boldsymbol{\Gamma}$ se dice que tiene distribución estacionaria $\boldsymbol{\delta}$ si $\boldsymbol{\delta\Gamma} = \boldsymbol{\delta}$ y $\boldsymbol{\delta 1} = 1$. El primer supuesto hace alusión a la estacionariedad y el segundo es un requerimiento de la distribución de probabilidad.

Mientras $\boldsymbol{u}(t+1) = \boldsymbol{u}(t)\boldsymbol{\Gamma}$, una cadena de Markov que empieza desde su distribución estacionaria continuará teniendo dicha distribución a través del tiempo, y este proceso se conoce como una cadena de Markov estacionaria. Por lo tanto, se está asumiendo mucho más que homogeneidad, ya que esta propiedad no es suficiente para garantizar estacionariedad debido a que el adjetivo estacionario es atribuido a aquellas cadenas de Markov homogéneas que adicionalmente tienen una distribución inicial $\boldsymbol{u}(1)$ estacionaria y por lo tanto son procesos estacionarios.

Adicionalmente una Cadena de Markov (homogénea, discreta, finita estado-espacio) irreducible tiene una única distribución estacionaria estrictamente positiva. Si se quiere adicionar la restricción de aperiodicidad, la única distribución que satisface este requerimiento es precisamente la distribución estacionaria, como se puede ver en (Feller, 1957). El vector $\boldsymbol{\delta}$ con valores no negativos es una distribución estacionaria de una cadena de Markov con matriz de transición de probabilidad $\boldsymbol{\Gamma}$ si y solo si:

$$\boldsymbol{\delta}(\boldsymbol{I}_m - \boldsymbol{\Gamma} + \boldsymbol{U}) = \mathbf{1}$$

donde $\mathbf{1}$ es un vector de unos, $\boldsymbol{I}_m$ es una matriz identidad $m \times m$ y $\boldsymbol{U}$ es una matriz de unos $m \times m$; así mismo, se puede llegar a una distribución estacionaria eliminando una de las ecuaciones del sistema, específicamente $\boldsymbol{\delta\Gamma} = \delta$ y reemplazándola por $\sum_i \delta_i = 1$.

2.2.2 Función de Auto-correlación

Asumiendo que se tiene un set de estados numéricos y no categóricos (letras o símbolos) representados por C_t asumiendo estacionariedad e irreductibilidad, se puede obtener la función de auto-correlación (ACF) de C_t de la siguiente forma:

1. Definiendo $\boldsymbol{v} = (1, \ldots, m)$ y $\boldsymbol{V} = \text{diag}(1, \ldots, m)$, se tiene para todo k no negativo:
$$\text{Cov}(C_t, C_{t+k}) = \boldsymbol{\delta V \Gamma}^k \boldsymbol{v'} - (\boldsymbol{\delta v'})^2, \tag{2.3}$$

2. Si la matriz $\boldsymbol{\Gamma}$ es diagonalizable y sus valores propios son denotados como $\omega_1, \ldots, \omega_m$, entonces $\boldsymbol{\Gamma}$ puede ser escrita como:
$$\boldsymbol{\Gamma} = \boldsymbol{U\Omega U}^{-1},$$

donde $\boldsymbol{\Omega}$ es la diag$(1, \omega_2, \omega_3, \ldots, \omega_m)$ y las columnas de $\boldsymbol{U}$ son los valores propios de $\boldsymbol{\Gamma}$. Se tiene:

$$\begin{aligned}\text{Cov}(C_t, C_{t+k}) &= \boldsymbol{\delta V U \Omega^k U^{-1} v'} - (\boldsymbol{\delta v'})^2 \\ &= \boldsymbol{a \Omega^k b'} - a_1 b_1 \\ &= \sum_{i=2}^{m} a_i b_i \omega_i^k,\end{aligned}$$

donde $\boldsymbol{a} = \boldsymbol{\delta V U}$ y $\boldsymbol{b'} = \boldsymbol{U}^{-1}\boldsymbol{v'}$. Siendo $\text{Var}(C_t) = \sum_{i=2}^{m} a_i b_i$ para todo k no negativo se tiene:

$$\rho(k) \equiv \text{Corr}(C_t, C_{t+k}) = \frac{\sum_{i=2}^{m} a_i b_i \omega_i^k}{\sum_{i=2}^{m} a_i b_i}, \tag{2.4}$$

lo cual sería un promedio ponderado por las $k-$ésimas potencias de los valores propios $\omega_1, \ldots, \omega_m$ y un término similar a la función de auto-correlación de un proceso Gausiano de orden $m-1$. Un ejemplo es el caso cuando $m = 2$, esto implica que $\rho(k) = \rho(1)^k$ para todo k no negativo y que $\rho(1)$ es el valor propio distinto a uno de $\boldsymbol{\Gamma}$.

2.2.3 Estimación de las Probabilidades de Transición

Si se tiene la realización de una cadena de Markov y se desean estimar las probabilidades de transición, la aproximación más sencilla es hallar las frecuencias de transición y estimar las probabilidades de transición relativas a la frecuencia total.

Supongase que se quiere estimar $m^2 - m$ parámetros γ_{ij}, con $i \neq j$, de una cadena de Markov $\{C_t\}$ con m estados que tiene como realización $c_1, c_2, \ldots, c_T$. La función de verosimilitud condicionada a la primera observación es:

$$L = \prod_{i=1}^{m} \prod_{j=1}^{m} \gamma_{ij}^{f_{ij}},$$

entonces, al aplicar logaritmo, la función de log-verosimilitud esta dada por:

$$l = \sum_{i=1}^{m} \left(\sum_{j=1}^{m} f_{ij} \log \gamma_{ij} \right) = \sum_{i=1}^{m} l_i(\cdots),$$

por lo tanto se puede maximizar l maximizando l_i por separado. Esto se lleva a cabo sustituyendo $1 - \sum_{k \neq i} \gamma_{ii}$, diferenciando l_i con respecto a los elementos fuera de la diagonal de la matriz de transición de probabilidad γ_{ij} e igualando esto a cero el resultado sería:

$$0 = \frac{-f_{ij}}{1 - \sum_{k \neq i} \gamma_{ii}} + \frac{f_{ij}}{\gamma_{ij}} = -\frac{f_{ii}}{\gamma_{ii}} + \frac{f_{ij}}{\gamma_{ij}}.$$

Analizando la anterior ecuación, a menos que el denominador sea igual a cero, el máximo de la función de verosimilitud esta dado por:

$$\gamma_{ii} = \frac{f_{ii}}{\sum_{j=1}^{m} f_{ij}} \quad \text{y} \quad \gamma_{ij} = \frac{f_{ij}\gamma_{ii}}{f_{ii}} = \frac{f_{ij}}{\sum_{j=1}^{m} f_{ij}}.$$

El estimador encontrado y denotado como $\hat{\gamma}_{ij} = \frac{f_{ij}}{\sum_{k=1}^{m} f_{ik}}$ para $i, j = 1, \ldots, m$ es un estimador empírico de la transición de probabilidad, por lo que es considerado un estimador de la función de verosimilitud condicional. Este estimador sigue satisfaciendo la restricción acerca de la suma de las filas de $\mathbf{\Gamma}$ suman uno.

El supuesto de estacionariedad de la cadena de Markov no fue usado en la derivación anterior. Si se deseara activar este supuesto, se debería utilizar la verosimilitud incondicional. La función de verosimilitud incondicional o su logaritmo, debe ser maximizada por medio de métodos numéricos, con las restricciones de no negatividad y la suma de las filas de la matriz de transición de probabilidad igual a uno.

2.3 Hidden Markov Model

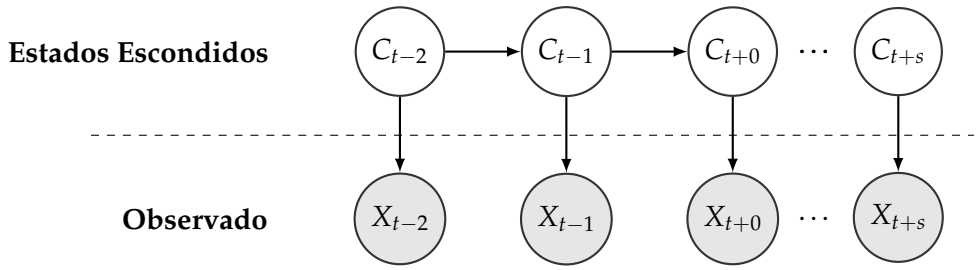
Los supuestos de los *mixture models* son muy fuertes, no permiten dependencia serial o correlación serial. Se relaja el supuesto de la independencia (serial) del proceso de parámetros. Una manera matemática para hacer lo anterior es asumir que se tiene una cadena de Markov.

Un HMM $\{X_t : t \in \mathbb{N}\}$ es un tipo particular de *mixture*. Con $\boldsymbol{X}^{(t)}$ y $\boldsymbol{C}^{(t)}$ representando los procesos o historias desde 1 hasta t, se puede resumir el modelo de la siguiente forma:

$$\mathbb{P}(C_t|\boldsymbol{C}^{(\boldsymbol{t-1})}) = \mathbb{P}(C_t|C_{t-1}), \quad t = 2, 3, \ldots \tag{2.5}$$

$$\mathbb{P}(X_t|\boldsymbol{X}^{(t-1)}, \boldsymbol{C}^{(t)}) = \mathbb{P}(X_t|C_t), \quad t \in \mathbb{N}. \tag{2.6}$$

El modelo consiste básicamente en dos partes: la primera, un no observado "proceso de parámetros" $\{C_t : t = 1, 2, \ldots\}$ que satisface la propiedad de Markov; y la segunda parte, un "proceso estado-dependiente" $\{X_t : t = 1, 2, \ldots\}$, en el cual la distribución de X_t depende solamente del estado C_t y no estados u observaciones previas.



Si la cadena de Markov $\{C_t\}$ tiene m estados, se considera a $\{X_t\}$ un HMM de m estados. Desde ahora se introducirá la notación que incluirá el caso de observaciones discretas y continuas. En el caso de observaciones discretas se define para $i = 1, 2, \ldots$:

$$p_i(x) = \mathbb{P}(X_t = x|C_t = i).$$

De allí se tiene a p_i como la función de probabilidad (mass) de X_t si la cadena de Markov esta en el tiempo t y estado i. Para el caso continuo p_i sería la función de densidad de probabilidad de X_t asociada con el estado i. Se tienen m distribuciones con p_i siendo las distribuciones estado-dependientes del modelo.

2.3.1 Distribuciones Marginales

Frecuentemente se necesita trabajar con la distribución marginal de X_t y a veces con distribuciones marginales de mayor orden, como e el caso de (X_t, X_{t+k}). Se asumirá que la cadena de Markov es homogénea pero no necesariamente estacionaria (como se menciono anteriormente la homogeneidad no implica necesariamente estacionariedad), pero para este caso se le atribuirá la estacionariedad a la cadena de Markov. Por efectos matemáticos solo se tendrá en cuenta las distribuciones estado-dependientes.

Distribuciones Univariadas

Para el caso de observaciones discretas del modelo de X_t, definiendo $u_i(t) = \mathbb{P}(C_t = i)$ para todo $t = 1, \ldots, T$, se tiene:

$$\begin{aligned}\mathbb{P}(X_t = x) &= \sum_{i=1}^{m} \mathbb{P}(C_t = i)\mathbb{P}(X_t = x | C_t = i) \\ &= \sum_{i=1}^{m} u_i(t) p_i(x).\end{aligned}$$

La anterior expresión se puede reescribir en términos de matrices de la siguiente forma:

$$\begin{aligned}\mathbb{P}(X_t = x) &= (u_1(t), \ldots, u_m(t)) \begin{pmatrix} p_1(x) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p_m(x) \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= \boldsymbol{u}(t)\boldsymbol{\mathcal{P}}(x)\mathbf{1}'.\end{aligned}$$

En la anterior expresión $\boldsymbol{\mathcal{P}}(x)$ es la matriz diagonal con elementos de su diagonal las m probabilidades $p_i(x)$. De la ecuación (2.2) se puede inferir que $\boldsymbol{u}(t) = \boldsymbol{u}(1)\boldsymbol{\Gamma}^{t-1}$ y por lo tanto:

$$\mathbb{P}(X_t = x) = \boldsymbol{u}(1)\boldsymbol{\Gamma}^{t-1}\boldsymbol{\mathcal{P}}(x)\mathbf{1}'. \tag{2.7}$$

La anterior ecuación, (2.7), sostiene el hallazgo que anteriormente se mencionó, si una cadena de Markov es homogénea no quiere decir que sea estacionaria. Si se asume que la cadena de Markov con la que se está trabajando es estacionaria, con distribución estacionaria $\boldsymbol{\delta}$, entonces (2.7) se simplificaría debido a que $\boldsymbol{\delta}\boldsymbol{\Gamma}^{t-1} = \boldsymbol{\delta}$ para todo $t \in \mathbb{N}$, entonces quedaría la siguiente expresión:

$$\mathbb{P}(X_t = x) = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}\mathbf{1}'. \tag{2.8}$$

Distribuciones Bivariada

La identificación de las distribuciones asociadas a un HMM son calculadas por medio de la distribución conjunta del set de variables aleatorias V_i:

$$\mathbb{P}(V_1, \ldots, V_n) = \prod_{i=1}^{n} \mathbb{P}(V_i | \text{pa}(V_i)), \tag{2.9}$$

donde el término $\text{pa}(V_i)$ es el set de parents de V_i para todo $i = 1, \ldots, n$. Ahora el caso de interés, si se tienen los elementos que componen el HMM, es decir, X_t, X_{t+k}, C_t

y C_{t+k} para toda k siendo un número entero positivo. Entonces se tiene que C_t no tiene *parents*, $\text{pa}(X_t) = \{C_t\}$, $\text{pa}(X_{t+k}) = \{C_{t+k}\}$, $\text{pa}(C_{t+k}) = \{C_t\}$ y por último $\text{pa}(C_{t+k}) = \{C_t\}$. Estos parents son dado el caso que que tenga solo las cuatro variables aleatorias $X_t, X_{t+k}, C_t, C_{t+k}$, entonces se tiene:

$$\mathbb{P}(X_t, X_{t+k}, C_t, C_{t+k}) = \mathbb{P}(C_t)\mathbb{P}(X_t|C_t)\mathbb{P}(C_{t+k}|C_t)\mathbb{P}(X_{t+k}|C_{t+k}),$$

por lo tanto se tendría la siguiente expresión:

$$\begin{aligned}
\mathbb{P}(X_t = v, X_{t+k} = w) &= \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbb{P}(X_t = v, X_{t+k} = w, C_t = i, C_{t+k} = j) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \underbrace{\mathbb{P}(C_t = i)}_{u_i(t)}\, p_i(v)\, \underbrace{\mathbb{P}(C_{t+k} = j|C_t = i)}_{\gamma_{ij}(k)}\, p_j(w) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} u_i(t)p_i(v)\gamma_{ij}(k)p_j(w).
\end{aligned}$$

Es necesario recordar que $\gamma_{ij}(k)$ es el elemento en la posición (i, j) de $\mathbf{\Gamma}^k$. Ahora escribiendo las sumatorias como el producto de matrices, la expresión se puede reescribir de la siguiente forma:

$$\mathbb{P}(X_t = v, X_{t+k} = w) = \boldsymbol{u}(t)\boldsymbol{\mathcal{P}}(v)\mathbf{\Gamma}^k\boldsymbol{\mathcal{P}}(w)\mathbf{1'}. \tag{2.10}$$

Así mismo es necesario recordar que si la cadena de Markov es estacionaria se puede reescribir como:

$$\mathbb{P}(X_t = v, X_{t+k} = w) = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(v)\mathbf{\Gamma}^k\boldsymbol{\mathcal{P}}(w)\mathbf{1'}. \tag{2.11}$$

Para generalizar, en los casos de distribuciones marginales de orden superior y que son estacionaria, la fórmula que representa la distribución para k y l siendo enteros positivos es:

$$\mathbb{P}(X_t = v, X_{t+k} = w, X_{t+k+l} = z) = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(v)\mathbf{\Gamma}^k\boldsymbol{\mathcal{P}}(w)\mathbf{\Gamma}^l\boldsymbol{\mathcal{P}}(z)\mathbf{1'}.$$

2.3.2 Momentos

El caso general, donde no se asume estacionariedad, se tiene:

$$\mathbb{E}(X_t) = \sum_{i=1}^{m} \mathbb{E}(X_t|C_t = i)\mathbb{P}(C_t = i) = \sum_{i=1}^{m} u_i(t)\mathbb{E}(X_t|C_t = i),$$

pero para el caso en que la estacionariedad está presente, la anterior ecuación se reduce a:

$$\mathbb{E}(X_t) = \sum_{i=1}^{m} \delta_i\mathbb{E}(X_t|C_t = i).$$

Este resultado se mantiene para cualquier función $g(\cdot)$ para la cual la esperanza del estado-dependiente exista. Retomando, la expresión estaría dada por:

$$\mathbb{E}(g(X_t)) = \sum_{i=1}^{m} \delta_i\mathbb{E}(g(X_t)|C_t = i),$$

y para encontrar la $\mathbb{E}(g(X_t, X_{t+k}))$ el procedimiento es similar, se obtiene:

$$\mathbb{E}(g(X_t, X_{t+k})) = \sum_{i,j=1}^{m} \delta_i \gamma_{ij}(k) \mathbb{E}(g(X_t, X_{t+k}) | C_t = i, C_{t+k} = j).$$

Estas expresiones son útiles en el sentido de calcular los distintos momentos de interés como covarianzas y correlaciones.

2.3.3 Función de verosimilitud

Es necesario identificar cual es la función de verosimilitud L_T de T observaciones $x_1, \ldots, x_T$ suponiendo que son generadas por un HMM de $m-$estados. El objetivo es encontrar la probabilidad L_T de observar la secuencia de observaciones, calculadas bajo el HMM mencionado, el cual tiene distribución inicial $\boldsymbol{\delta}$ y matriz de transición de probabilidad $\boldsymbol{\Gamma}$ para la cadena de Markov asociada, de igual forma, se tiene el estado-dependiente de la función de probabilidad (densidad) denotado como p_i. Anteriormente se asumía que $\boldsymbol{\delta}$ es la distribución estacionaria asociada a $\boldsymbol{\Gamma}$, pero en general no es necesario realizar dicho supuesto.

Proposición 1

La función de verosimilitud esta dada por:

$$L_T = \boldsymbol{\delta} \boldsymbol{\mathcal{P}}(x_1) \boldsymbol{\Gamma} \boldsymbol{\mathcal{P}}(x_2) \cdots \boldsymbol{\Gamma} \boldsymbol{\mathcal{P}}(x_T) \mathbf{1}'. \tag{2.12}$$

Si $\boldsymbol{\delta}$, la distribución de C_1, es la distribución estacionaria de la cadena de Markov, se tiene:

$$L_T = \boldsymbol{\delta} \boldsymbol{\Gamma} \boldsymbol{\mathcal{P}}(x_1) \boldsymbol{\Gamma} \boldsymbol{\mathcal{P}}(x_2) \cdots \boldsymbol{\Gamma} \boldsymbol{\mathcal{P}}(x_T) \mathbf{1}'. \tag{2.13}$$

Prueba: Proposición 1

Para el caso de observaciones discretas, primero es valido notar que:

$$L_T = \mathbb{P}(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}) = \sum_{c_1,c_2,\ldots,c_T=1}^{m} \mathbb{P}(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}, \boldsymbol{C}^{(T)} = \boldsymbol{c}^{(T)}),$$

y gracias a la ecuación (2.9) se tiene:

$$\mathbb{P}(\boldsymbol{X}^{(T)}, \boldsymbol{C}^{(T)}) = \mathbb{P}(C_1) \prod_{k=2}^{T} \mathbb{P}(C_k|C_{k-1}) \prod_{k=1}^{T} \mathbb{P}(X_k|C_k). \tag{2.14}$$

Por lo tanto:

$$\begin{aligned} L_t &= \sum_{c_1,\ldots,c_T=1}^{m} (\delta_{c_1}\gamma_{c_1,c_2}\gamma_{c_2,c_3}\cdots\gamma_{c_{T-1},c_T}) \left(p_{c_1}(x_1)p_{c_2}(x_2)\cdots p_{c_T}(x_T)\right) \\ &= \sum_{c_1,\ldots,c_T=1}^{m} \delta_{c_1} p_{c_1}(x_1)\gamma_{c_1,c_2} p_{c_2}(x_2)\gamma_{c_2,c_3}\cdots\gamma_{c_{T-1},c_T}\cdots p_{c_T}(x_T) \\ &= \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(x_1)\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_2)\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_2)\cdots\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_T)\mathbf{1}', \end{aligned}$$

la anterior ecuación es la misma que la ecuación (2.12). Si $\boldsymbol{\delta}$ es la distribución estacionaria de la cadena de Markov, se tiene que:

$$\boldsymbol{\delta}\boldsymbol{\mathcal{P}}(x_1) = \boldsymbol{\delta}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_1),$$

por lo tanto la ecuación (2.13) tiene un $\boldsymbol{\Gamma}$ extra, lo cual permite facilitar las operaciones computacionales, porque es meter todo dentro de un ciclo sin condiciones especiales. □

Un factor importante que influye en porque escribir la función de verosimilitud en términos matriciales, es el "forward algorithm". Su calculo en forma recursiva juega un papel clave no solo en la estimación y evaluación de la función de máxima verosimilitud, también en el pronostico y validación del modelo. Teniendo las ecuaciones (2.12) y (2.13) el cálculo de la función de verosimilitud es computacionalmente más eficiente en comparación a la serie de sumatorias expresadas anteriormente.
Para desarrollar el "forward algorithm" se define el vector $\boldsymbol{\alpha}_t$, para $t = 1, 2, \ldots, T$ igual a:

$$\boldsymbol{\alpha}_t = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(x_1)\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_2)\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_3)\cdots\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_t) = \boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_1)\prod_{s=2}^{t}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_s), \tag{2.15}$$

Entonces la función de verosimilitud se puede reesxpresar como:

$$L_T = \boldsymbol{\alpha}_T\mathbf{1}' \quad \text{siendo} \quad \boldsymbol{\alpha}_T = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_t) \quad \text{para} \quad t \geq 2.$$

Ahora utilizando la ecuación **??** se tiene:

$$\boldsymbol{\alpha}_1 = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(x_1);$$

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_t) \quad \text{para} \quad t = 2, 3, \ldots, T;$$

$$L_T = \boldsymbol{\alpha}_T\mathbf{1}'$$

Para poder llegar a la ecuación (2.13) se calculan:

$$\boldsymbol{\alpha}_0 = \boldsymbol{\delta};$$
$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_t) \quad \text{para} \quad t = 2, 3, \ldots, T;$$
$$L_T = \boldsymbol{\alpha}_T \mathbf{1}'$$

Los elementos que componen el vector $\boldsymbol{\alpha}_t$ se les denomina "forward probabilities". El $j-$ésimo elemento de éste vector esta dado por $\mathbb{P}(\boldsymbol{X}^{(t)} = \boldsymbol{x}^{(t)}, C_t = j)$.

2.3.4 Propiedad de Markov: HMM

No todos los HMM satisfacen la propiedad de Markov, sin embargo existen condiciones que generalizan la satisfacción de esta propiedad por parte de los HMM, los estudios que abordan este tema hacen uso de metodologías asociadas al filtrado recursivo de las cadenas de Markov subyacentes al HMM (para más información consultar (Spreij, 2001)).

2.4 EM Algorithm

Una de las metodologías más comunes para encontrar la máxima verosimilitud del HMM es el algoritmo EM. Éste también es conocido como el algoritmo de Baum-Welch, el cual fue diseñado para estimar los parámetros de un HMM cuya cadena de Markov se asume homogenea pero no estacionaria (no se cumple $\boldsymbol{\delta}\boldsymbol{\Gamma} = \boldsymbol{\delta}$).

2.4.1 Probabilidades Forward y Backward

En la sección anterior se definió $\boldsymbol{\alpha}_t$ en la ecuación (2.15), cuyos elementos describen las probabilidades forward. Para poder realizar la estimación se debe denotar el vector de probabilidades backward, dado por $\boldsymbol{\beta}_t$:

$$\boldsymbol{\beta}_t' = \boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_{t+1})\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_{t+2})\cdots\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_T)\mathbf{1}' = \left(\prod_{s=t+1}^{T}\boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_s)\right)\mathbf{1}', \tag{2.16}$$

donde para $t = T$ se tiene $\boldsymbol{\beta}_T = 1$. Los elementos del vector $\boldsymbol{\beta}_t$ pueden ser interpretados como las probabilidades condicionales $\mathbb{P}(X_{t+1} = x_{t+1}, \ldots, X_T = x_T | C_t = j)$, entonces para $t = 1, \ldots, T$ se tiene:

$$\alpha_t(j)\beta_t(j) = \mathbb{P}(\boldsymbol{X}^{(T)} = \boldsymbol{x}^{(T)}, C_t = j) \tag{2.17}$$

Los componentes de la ecuación (2.17) serán justificados por medio de una serie de proposiciones más adelante.

Forward Probabilities

De la definición de $\boldsymbol{\alpha}_t$ para todo $i = 1, \ldots, T-1$, se tendría $\alpha_{t+1} = \boldsymbol{\alpha}_t\boldsymbol{\Gamma}\mathcal{P}(x_{t+1})$, o tambien como:

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{m}\alpha_t(i)\gamma_{ij}\right)p_j(x_{t+1}).$$

Teniendo en cuenta la anterior ecuación se tiene la siguiente proposición.

Proposición 2

Para todo $t = 1, \ldots, T$ y $j = 1, \ldots, m$ se tiene:

$$\alpha_t(j) = \mathbb{P}(\boldsymbol{X}^{(t)} = x^{(t)}, C_t = j)$$

Prueba: Proposición 2

La siguiente prueba se realizará por inducción. Luego, si $\boldsymbol{\alpha}_1 = \boldsymbol{\delta}\boldsymbol{\mathcal{P}}(x_1)$, se tiene:

$$\alpha_1(j) = \delta_j p_j(x_1) = \mathbb{P}(C_1 = j)\mathbb{P}(X_1 = x_1 | C_1 = j),$$

entonces $\alpha_1(j) = \mathbb{P}(X_1 = x_1, C_1 = j)$ cuando $t = 1$. Ahora para generalizar se tiene que cuando $t \in \mathbb{N}$, entonces para $t + 1$ se tiene:

$$\begin{aligned}
\alpha_{t+1}(j) &= \sum_{i=1}^{m} \alpha_t(i)\gamma_{ij}p_j(x_{t+1}) \\
&= \sum_i \mathbb{P}(\boldsymbol{X}^{(t)} = x^{(t)}, C_t = i)\mathbb{P}(C_{t+1} = j | C_t = i)\mathbb{P}(X_{t+1} = x_{t+1} | C_{t+1} = j) \\
&= \sum_i \mathbb{P}(\boldsymbol{X}^{(t+1)} = x^{(t+1)}, C_t = i, C_{t+1} = j) \\
&= \mathbb{P}(\boldsymbol{X}^{(t+1)} = x^{(t+1)}, C_{t+1} = j)
\end{aligned}$$

□

Dado que que $\alpha_t(i)$ es la probabilidad conjunta de las observaciones en t y el estado escondido C_t será pequeño para valores grandes de t. La razón de ser consideradas *forward probabilities* es que las probabilidades son calculadas por medio de una recursión adelante en el tiempo.

Backward Probabilities

Se tenía la definición de β_t para $t = 1, 2, \ldots, T-1$ como $\boldsymbol{\beta}_t' = \boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_{t+1})\boldsymbol{\beta}_{t+1}'$

Proposición 3

Para todo $t = 1, \ldots, T-1$ y toda $i = 1, \ldots, m$, se tiene:

$$\beta_t(i) = \mathbb{P}(X_{t+1} = x_{t+1}, X_{t+2} = x_{t+2}, \ldots, X_T = x_T | C_t = i)$$

Se tiene que $\mathbb{P}(C_t = i) > 0$ y se puede escribir la anterior ecuación de una forma compacta:

$$\beta_t(i) = \mathbb{P}(\boldsymbol{X}_{t+1}^{T} = x_{t+1}^{T} | C_t = i)$$

Se puede observar que $\beta_t(i)$ es una probabilidad condicional, a diferencia de $\alpha_t(i)$, la cual es una probabilidad conjunta. $\beta_i(t)$ es la probabilidad que las observaciones sean $x_{t+1}, \ldots, x_T$, dado que la cadena de Markov esta en el estado i en el tiempo t. Así mismo son llamadas *backward probabilities* gracias a que su estimación se realiza por medio de una recursión hacia atrás en el tiempo.

Prueba: Proposición 3

La siguiente prueba se realizará por inducción. Luego, se tienen las siguientes ecuaciones:

$$\mathbb{P}(\boldsymbol{X}_{t+1}^T|C_{t+1}) = \mathbb{P}(X_{t+1}|C_{t+1})\mathbb{P}(\boldsymbol{X}_{t+2}^T|C_{t+1})$$
$$\mathbb{P}(\boldsymbol{X}_{t+1}^T|C_{t+1}) = \mathbb{P}(\boldsymbol{X}_{t+1}^T|C_t, C_{t+1})$$

Para establecer la validez para $t = T-1$ es necesario notar $\beta'_{T-1} = \boldsymbol{\Gamma}\boldsymbol{\mathcal{P}}(x_T)\mathbf{1}'$ de forma que:

$$\beta_{T-1}(i) = \sum_j \mathbb{P}(C_T = j|C_{T-1} = i)\mathbb{P}(X_T = x_T|C_T = j),$$

por lo que se puede tomar las ecuaciones iniciales y llegar a:

$$\begin{aligned}\mathbb{P}(C_T|C_{T-1})\mathbb{P}(X_T|C_T) &= \mathbb{P}(C_t|C_{T-1})\mathbb{P}(X_t|C_{T-1}, C_T)\\ &= \mathbb{P}(X_T, C_{T-1}, C_T)/\mathbb{P}(C_{T-1}).\end{aligned}$$

Ahora se necesitan realizar las sustituciones en las anteriores ecuaciones y el procedimiento esta representado por las siguientes ecuaciones:

$$\begin{aligned}\beta_{T-1}(i) &= \frac{1}{\mathbb{P}(C_{T-1} = i)}\sum_j \mathbb{P}(X_T = x_T, C_{T-1} = i, C_T = j)\\ &= \mathbb{P}(X_T = x_T, C_{T-1} = i)/\mathbb{P}(C_{T-1} = i)\\ &= \mathbb{P}(X_T = x_T|C_{T-1} = i).\end{aligned}$$

Por inducción solo falta probar la validez de t, por lo tanto:

$$\beta_t(i) = \sum_j \gamma_{ij}\mathbb{P}(X_{t+1} = x_{t+1}|C_{t+1} = j)\mathbb{P}(\boldsymbol{X}_{t+2}^T = x_{t+2}^T|C_{t+1} = j)$$

□

y se tiene tambien:

$$\mathbb{P}(\boldsymbol{X}_{t+1}^T|C_t, C_{t+1}) = \mathbb{P}(X_{t+1}|C_{t+1})\mathbb{P}(\boldsymbol{X}_{t+2}^T|C_{t+1})$$

y al sustituir de igual forma que en el anterior caso se tiene lo siguiente:

$$\begin{aligned}\beta_t(i) &= \sum_j \mathbb{P}(C_{t+1} = j|C_t = i)\mathbb{P}(\boldsymbol{X}_{t+1}^T|C_t = i, C_{t+1} = j)\\ &= \frac{1}{\mathbb{P}(C_t = i)}\sum_j \mathbb{P}(\boldsymbol{X}_{t+1}^T = x_{t+1}^T, C_t = i, C_{t+1} = j)\\ &= \mathbb{P}(\boldsymbol{X}_{t+1}^T = x_{t+1}^T, C_t = i)/\mathbb{P}(C_t = i),\end{aligned}$$

la cual es una probabilidad condicional, por ende los requerimientos están satisfechos. □

Algoritmo EM

Previamente se especificó que la cadena de Markov que compone el HMM no es observada, por ello, la estimación de los parámetros del modelo se realiza tratando la secuencia de estados representados por la cadena como data faltante y se emplea el algoritmo EM con el fin de encontrar el máximo de la función de verosimilitud. Los primeros trabajos que se aproximan al algoritmo EM fueron los de Leonard Baum y Lloyd Welch.

El algoritmo EM es un método iterativo para encontrar la máxima verosimilitud cuando existen problemas de completitud en los datos, explotando el hecho que la función de verosimilitud con datos completos (CDLL por sus siglas en ingles: Complete-data log-likelihood), la cual es relativamente sencilla de maximizar incluso si la data observada no esta disponible o hace falta. La CDLL hace referencia a la función de log-verosimilitud que conduce a los parámetros de interés $\boldsymbol{\theta}$, basado tanto en los datos observados como los faltantes.

En resumen, el algoritmo tiene como primer paso asignar unos valores iniciales para los parámetros $\boldsymbol{\theta}$ que se desean estimar. Los siguientes dos pasos se deben repetir iterativamente hasta que se cumpla un criterio de convergencia:

- **Paso E**: Consiste en calcular la esperanza condicional de los datos faltantes dadas las observaciones y el estimado parcial de los parámetros $\boldsymbol{\theta}$, es decir, calcular la esperanza condicional de aquellas funciones de la data faltante que hacen parte de la CDLL.
- **Paso M**: Este paso busca maximizar con respecto a los parámetros $\boldsymbol{\theta}$ la CDLL, ésta función estará compuesta por las funciones correspondientes de la data disponible y las funciones relacionadas con la data faltante serán reemplazadas por las esperanzas calculadas en el paso anterior.

Algoritmo EM: Aplicación para los HMM

Para el caso de los HMM existe una alternativa que permite simplificar algunos procedimientos matemáticos posteriores que consiste en representar la secuencia de estados $c_1, c_2, \ldots, c_T$ seguidos por la cadena de Markov compuesta por variables aleatorias cero-uno definidas de la siguiente forma:

$$u_j(t) = 1 \quad \text{si y solo si} \quad c_t = j \quad (t = 1, 2, \ldots, T)$$

así mismo es necesario definir:

$$v_{jk}(t) = 1 \quad \text{si y solo si} \quad c_{t-1} = j \quad \text{y} \quad c_t = k \quad (t = 1, 2, \ldots, T)$$

Con esta notación, la CDLL de un HMM (la función de log-verosimilitud de las observaciones $x_1, x_2, \ldots, x_T$ mas los datos faltantes $c_1, c_2, \ldots, c_T$) se representaría por la siguiente ecuación:

$$\log\left(\mathbb{P}\left(x^{(T)}, c^{(T)}\right)\right) = \log\left(\delta_{c_1}\prod_{t=2}^{T}\gamma_{c_{t-1},c_t}\prod_{t=1}^{T}p_{c_t}(x_t)\right) \tag{2.18}$$

$$= \log\delta_{c_1} + \sum_{t=2}^{T}\log\gamma_{c_{t-1},c_t} + \sum_{t=1}^{T}\log p_{c_t}(x_t) \tag{2.19}$$

$$= \sum_{j=1}^{m}u_j(1)\log\delta_j + \sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{t=2}^{T}v_{jk}(t)\right)\log_{jk} + \sum_{j=1}^{m}\sum_{t=1}^{T}u_j(t)\log p_j(x_t) \tag{2.20}$$

es decir $\boldsymbol{\delta}$ es interpretado como la distribución inicial de la cadena de Markov. De ahora en adelante es necesario denotar a $L_t = \alpha_t\beta_t$. El algoritmo EM para los HMM procede de la siguiente forma:

- **Paso E**: Reemplazar v_{jk} y $u_j(t)$ por la esperanza condicional cuando están dadas las observaciones $\boldsymbol{x}^{(T)}$ y los parámetros iniciales:

$$\hat{u}_j(t) = \mathbb{P}(C_t = j|\boldsymbol{x}^{(T)}) = \alpha_t(j)\beta_t(j)/L_T \tag{2.21}$$

  y

$$\hat{v}_{jk}(t) = \mathbb{P}(C_{t-1} = j, C_t = k|\boldsymbol{x}^{(T)}) = \alpha_{t-1}\gamma_{jk}p_k(x_t)\beta_t(k)/L_T \tag{2.22}$$

  Es notable que para poder realizar este paso es necesario estimar las *forward probabilities* para un HMM que no se asume estacionario bajo la cadena de Markov $\{C_t\}$. Así mismo es valido aclarar que las *backward probabilities* no son afectadas bajo el supuesto de estacionariedad o la negación de este supuesto sobre $\{C_t\}$.

- **Paso M**: Al reemplazar $v_{jk}(t)$ y $u_j(t)$ por $\hat{v}_{jk}(t)$ y $\hat{u}_j(t)$ y maximizar la CDLL con respecto a los parámetros: la distribución inicial $\boldsymbol{\delta}$ y $\boldsymbol{\Gamma}$ (t.p.m) y los parámetros de las distribuciones que dependen de los estados. Esto permite observar que este paso se divide en tres procedimientos, o tres maximizaciones (de los parámetros) gracias a que el primer término de la función objetivo depende de de parámetros independientes. Como tal la maximización al manejarla dependiendo de que parámetros están compuestos quedaría dividida de la siguiente forma:

  1. $\sum_{j=1}^{m}\hat{u}_j(1)\log\delta_j$ con respecto a $\boldsymbol{\delta}$.
  2. $\sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{t=2}^{T}\hat{v}_{jk}(t)\right)\log\gamma_{jk}$ con respecto a $\boldsymbol{\Gamma}$.
  3. $\sum_{j=1}^{m}\sum_{t=1}^{T}\hat{u}_j(t)\log p_j(x_t)$ con respecto a los parámetros estado-dependientes.

  La solución a estas maximizaciones esta dado por:

  1. $\delta_j = \hat{u}_j(1)\left(\sum_{j=1}^{m}\hat{u}_j(1)\right)^{-1} = \hat{u}_j(1)$.
  2. $\delta_{jk} = f_{ik}\left(\sum_{k=1}^{m}f_{jk}\right)^{-1}$ donde $f_{jk} = \sum_{t=2}^{T}\hat{v}_{jk}(t)$.
  3. La maximización del ítem 3 depende de la naturaleza de la distribución de los parámetros estado-dependientes, se trata como un problema de estimación de máxima verosimilitud.

2.5 Optimización de Portafolio: Modelo de Black Litterman

El modelo de optimización de portafolio propuesto por Fischer Black y Robert Litterman (Black and Litterman, 1990) es una versión actualizada o mejorada del modelo de media-varianza de Markowitz, ya que considera los siguientes aspectos: hay n activos, con capitalizaciones m_i con $i = 1, \ldots, n$. La capitalización de mercado es igual al número de títulos o unidades del activo disponibles en el mercado por su respectivo precio. Las ponderaciones de mercado de los n activos estarán dadas por el vector $\boldsymbol{w}$, de modo que la ponderación de los activos esta dada por:

$$\boldsymbol{w}_i = \frac{m_i}{\sum_{i=1}^{n} m_i}$$

Adicionalmente es necesario calcular un coeficiente de aversión al riesgo (λ), el cual se calcula por medio de la siguiente formula (Grinold and Kahn, 2000) :

$$\lambda = \frac{r_m - r_f}{\sigma_m^2}$$

donde r_m es el retorno de mercado; r_f es la tasa libre de riesgo y σ_m^2 es la varianza del retorno del mercado. El exceso de retornos implícito de equilibrio estará dado por:

$$\Pi = \lambda \Sigma \boldsymbol{w}$$

A Π se le denomina retornos implícitos de equilibrio debido a que si los precios de los activos se ajustan hasta que los retornos esperados sean iguales a lo que consideran los inversionistas, haciendo el supuesto que todos tienen la misma expectativa del mercado, esos ajustes hacen que la demanda iguale a la oferta.

El vector que representa los excesos de retorno $\boldsymbol{r} = (r_1, \ldots, r_n)$, es igual al retorno de cada activo menos la tasa libre de riesgo. Los excesos de retorno se asumen que siguen una distribución normal de forma:

$$\boldsymbol{r} \sim \mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Así mismo se asume que $\boldsymbol{\mu}$ tiene una distribución de probabilidad, de modo que la primera distribución representa el equilibrio:

$$\boldsymbol{\mu} \sim \mathrm{N}(\boldsymbol{\Pi}, \tau\boldsymbol{\Sigma})$$

donde τ es una constante que se puede interpretar como el grado de incertidumbre con respecto a la precisión con la que es estimado o calculado $\boldsymbol{\Pi}$. Este valor siempre se encuentra entre 0 y 1, generalmente 0.01 y 0.05 ya que la incertidumbre sobre la media debe ser menor a la incertidumbre de la variable.

La segunda distribución es la que representa las expectativas del inversionista sobre los retornos del mercado. Se tiene un conjunto de $k-$expectativas representadas con relaciones lineales. La expectativa se plantea como que el retorno esperado de un portafolio p_k tiene una distribución normal con promedio q_k y una desviación estándar representada por ω_k. Las $k-$expectativas con los correspondientes retornos esperados se expresan como:

$$\boldsymbol{P}' = [p_1, \ldots, p_k]_{k \times n}$$
$$\boldsymbol{Q}' = [q_1, \ldots, q_k]$$

Entonces $\boldsymbol{P}$ es la matriz que selecciona los activos que hacen parte de una expectativa y $\boldsymbol{Q}$ es el vector de expectativas, contiene el retorno esperado para cada activo que compone el portafolio.

Mediante $\boldsymbol{P}$ y $\boldsymbol{Q}$, se utiliza un esquema de ponderación por capitalización de mercado para determinar cada uno de los elementos de $\boldsymbol{P}$ diferentes de cero, en vez de utilizar el esquema de igual ponderación (Idzorek, 2004). Así, la ponderación individual de cada activo es proporcional a la capitalización de mercado del activo dividida por la capitalización del mercado total de los activos con cualquiera que sea su desempeño (positivo o negativo). La manera de expresar las expectativas es la siguiente:

$$\boldsymbol{P} \cdot \boldsymbol{\mu} = \boldsymbol{Q} + \varepsilon$$

donde ε es el vector aleatorio con media cero y matriz diagonal de covarianzas (lo cual se interpreta como la independencua de los componentes entre sí) $\boldsymbol{\Omega}$ normalmente distribuido, entonces:

$$\boldsymbol{P} \cdot \boldsymbol{\mu} \sim \mathrm{N}(\boldsymbol{Q}, \boldsymbol{\Omega})$$

Ahora, dadas las matrices $\boldsymbol{P}$ y $\boldsymbol{Q}$ se pueden calcular los **excesos de retorno implícitos ajustados** (views) y la matriz de de covarianza del error (incertidumbre):

$$\boldsymbol{\Omega} = \tau \cdot \boldsymbol{P} \cdot \Sigma \cdot \boldsymbol{P}$$
$$\Pi^* = \left[(\tau\Sigma)^{-1} + \boldsymbol{P}'\boldsymbol{\Omega}^{-1}\boldsymbol{P}\right] \cdot \left[(\tau\Sigma)^{-1}\Pi + \boldsymbol{P}'\boldsymbol{\Omega}^{-1}\boldsymbol{Q}\right]$$

donde $\boldsymbol{\Omega}$ esta dada por:

$$\boldsymbol{\Omega} = \begin{bmatrix} \omega_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \omega_k \end{bmatrix} = \begin{bmatrix} (p_1 \Sigma p_1')\tau & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (p_k \Sigma p_k')\tau \end{bmatrix}$$

Mientras ω_{ii} sea mayor, significa que existe un grado de confianza menor en los retornos esperados de $\boldsymbol{Q}$.El segundo término es la forma de calibrar el modelo (He and Litterman, 1999). Al encontrar los retornos implícitos ajustados por los views del inversionista se encuentra el portafolio tangente por medio de la optimización tradicional.

2.6 Pairs Trading: Conceptos

Pairs trading es una estrategia que busca aprovechar las características de diversos pares de activos con el fin de encontrar oportunidades de inversión dado que este par de activos se comportan con trayectoria similar a través del tiempo. Esta estrategia asume que el spread entre los dos activos cumple el supuesto de reversión a la media, lo cual intuitivamente es afirmar que las trayectorias de los dos activos mantienen una distancia que estadisticamente converge a un valor medio.

Existen dos formas de abordar la selección de activos dependiendo del criterio, según la distancia (Gatev, Goetzmann, and Rouwenhorst, 2006) o cointegración (Harris and Sollis, 2003). Dependiendo del enfoque, se realiza la selección de los pares de activos que cumplan los criterios necesarios para considerarse idóneos para aplicar la estrategia. Posteriormente se realizan unos cálculos estadísticos, de tal forma que sea posible tener un criterio de compra o venta sobre los activos.

El criterio de compra o venta está dado por una medida conocida como z-score, la cual es la estandarización del precio en t dada una media y desviación estándar atada a una ventana de tiempo.

Un claro ejemplo de esta estrategia sería suponer que A y B son dos activos que cumplen alguno de los dos criterios anteriormente mencionados, por lo tanto son idóneos par aplicar la estrategia. El primer paso para ejecutar la estrategia es acceder a los datos históricos de ambos activos y construir el proceso estocástico del spread $S_{t,[A,B]}$. A este nuevo activo compuesto por A y B es necesario calcularle la media y la desviación estándar para una ventana de tiempo W, posteriormente se estandarizan las observaciones para llegar al z-score del momento τ, $z(A,B)_\tau$y el criterio de decisión para generar ordenes de mercado esta representado por la siguiente función:

$$\text{order}(S_{t,[A,B]})_t = \begin{cases} \text{Si } z(A,B)_\tau > 1 \Rightarrow \text{vender } A \text{ y comprar B} \\ \text{Si } z(A,B)_\tau < -1 \Rightarrow \text{comprar } A \text{ y vender B} \end{cases} \tag{2.23}$$

Normalmente se compra y se vende la misma cantidad de unidades de ambos activos ya que se esta calculando el z-score a partir del spread de ambos activos 1:1, sin embargo existen metodologías que permiten hallar la cantidad de acciones deseables para cada activo que compone el par por medio de regresiones lineales.

Capítulo 3

Metodología y Resultados

El objetivo de este capítulo es construir un portafolio dinámico y rentable con la ayuda de diversas herramientas cuantitativas. Se hará uso de algoritmos de *machine learning* (DBSCAN y HMM), un modelo de asignación de activos (Modelo de Black-Litterman) y una estrategia de trading conocida como *pairs trading*.

El plan de trabajo puede ser descrito por los siguientes pasos:

1. Acceder a los datos: Quantopian.
2. Selección de Activos: DBSCAN Clustering y Test de Cointegración.
3. Construcción de los Views: HMM.
4. Asignación de Activos: Black-Litterman.
5. Implementación de la Estrategia: Pairs Trading.
6. Backtesting.

A través de este capitulo se encontrará la explicación de los procedimientos necesarios para la construcción de un algoritmo capaz de administrar un portafolio realizando trading de alta frecuencia en los activos seleccionados dadas unas pautas y condiciones.

3.1 Quantopian

Quantopian es un *hedge fund* que disponibiliza datos de acciones del mercado estadounidense con el fin de promover la investigación cuantitativa y desarrollo de algoritmos de inversión en mercado. La mayor parte de los datos utilizados en este trabajo fueron explotados desde la plataforma Quantopian, la cual es totalmente gratis y se accede al *cluster* por medio de un Jupyter Notebook (entorno de investigación), por lo tanto, el lenguaje de programación que se manejará es Python.

El *data warehouse* de Quantopian tiene información de más de 5000 compañías, bajo 600 distintas métricas desde el año 2002. Una ventaja que se tiene al trabajar con este repositorio de datos es la amplia gama de frecuencias en las cuales se pueden agregar o desagregar los datos. De igual forma Quantopian provee una capacidad de memoria eficiente para poder realizar los procedimientos eficientemente.

El primer procedimiento necesario para acceder a los datos es importar las librerías que permitirán hacer un llamado al universo de activos, interpretarlos y filtrarlos

por medio de unas condiciones deseadas para posteriormente poder realizar la consulta final. El siguiente cuadro de código contiene las librerías locales necesarias para empezar a interactuar con el ecosistema de Quantopian:

```
from quantopian.pipeline.data import morningstar
from quantopian.pipeline.filters.morningstar import Q1500US
from quantopian.pipeline import Pipeline
from quantopian.research import prices, symbols
from quantopian.research import run_pipeline
```

Ahora es necesario definir el universo de activos con los que se quiere trabajar. Uno de los criterios más deseados en modelos de alta frecuencia es la liquidez de los activos, por eso se trabajará con el universo de activos definido por Morningstar como `US 1500`, el cual contempla únicamente los 1500 activos con mayor liquidez en el mercado actualmente. Teniendo el universo de activos definido es necesario establecer que criterios o medidas se van a utilizar para realizar el primer filtro del universo de activos. Se hará uso de fundamentales tales como la capitalización de mercado, la industria a la cual pertenece la compañía y la salud financiera. Como se puede observar en el siguiente cuadro de código, la manera de hacer consultas de datos en Quantopian es por medio de `pipelines`:

```
study_date = "2019-04-01"
universe = Q1500US()

pipe = Pipeline(
    columns= {
        'Market Cap':morningstar.valuation.market_cap.latest.quantiles(5),
        'Industry': morningstar.asset_classification.
    morningstar_industry_group_code.latest,
        'Financial Health': morningstar.asset_classification.
    financial_health_grade.latest
    },
    screen=universe
)
```

Es necesario definir las condiciones sobre los fundamentales para saber que activos cumplen con estas. La primera condición es sobre a que sector de la industria pertenecen, se aceptarán todos los sectores industriales excepto conglomerados[1] y aquellas compañías que tengan datos disponibles respecto a su salud financiera.

El siguiente código realiza los filtros que se mencionaron previamente y se listan las compañías que cumplen con estas condiciones para realizar el query con el que se realizará la selección óptima de los activos por medio del clustering y test de cointegración. Se consulta únicamente precios de cierre con frecuencia diaria, la fecha de estudio será el 1 de Abril de 2019 y sus 150 meses previos[2].

```
res = res[res['Industry']!=31055]

res = res[res['Financial Health']!= None]

pricing = get_pricing(
```

[1]Se consideran conglomerados aquellas empresas que tengan fuentes de ingresos provenientes de más de tres filiales. No son deseables en la muestra ya que su comportamiento puede ser impactado por muchos factores y podrían llegar a generar relaciones espureas. Su código de industria es 31055.

[2]Es decir, desde el 1 de Noviembre de 2006 hasta el 1 de Abril de 2019

```
    symbols=res.index,
    fields='close_price',
    start_date=pd.Timestamp(study_date) - pd.DateOffset(months=150),
    end_date=pd.Timestamp(study_date),
    frecuency = 'day')
```

3.2 DBSCAN Clustering y Test de Cointegración

Al tener acceso a los datos de los 1498 activos que cumplen con los filtros previamente impuestos sobre los fundamentales, para identificar cuales son los stocks objetivo es necesario tener en cuenta que activos cumplen con las condiciones necesarias para implementar la estrategia de pairs trading. Como se mencionaba en el capítulo anterior (Marco Teórico - Pairs Trading) existen dos metodologías para identificar los pares de activos para implementar la estrategia, la distancia y la cointegración. Para efectos de este trabajo se utilizarán ambas metodologías, empezando por el criterio de distancia y posteriormente realizando un test de cointegración a los pares de series de precios de los activos que sean validados por el criterio de distancia.

Para el criterio de distancia se utilizará un algoritmo de clustering basado en la densidad de aplicaciones con ruido, conocido como DBSCAN. Este algoritmo trabaja identificando el número de clusters necesarios para agrupar cierto número mínimo de entidades -en este caso activos- con características homogéneas dado que se trabajará con datos ruidosos.

Posteriormente se realiza la descomposición de los datos en sus primeros componentes principales con el fin de acotar las dimensiones de estos, así mismo se realiza el reescalamiento de la información fundamental con el fin de proceder a implementar el algoritmo, como se puede observar en el siguiente cuadro de código:

```
N_PRIN_COMPONENTS = 50
pca = PCA(n_components=N_PRIN_COMPONENTS)
pca.fit(returns)

pca.components_.T.shape

X = np.hstack(
    (pca.components_.T,
     res['Market Cap'][returns.columns].values[:, np.newaxis],
     res['Financial Health'][returns.columns].values[:, np.newaxis])
)

print(X.shape)

X = preprocessing.StandardScaler().fit_transform(X)
print(X.shape)
```

Se establece 3 como mínimo número de activos que deben pertenecer al un cluster, ya que se necesitan por lo menos dos o más posibles combinaciones de activos para poner a prueba posteriormente. Al implementar el algoritmo se debe tener en cuenta la posible existencia de 409965 pares dentro de los datos disponibles, sin embargo al llevar a cabo el clustering se descubren 10 clusters, con un total de 1674 posibles pares para evaluar con el criterio de cointegración. El código que permitió llevar a cabo el procedimiento previo fue el siguiente:

```
clf = DBSCAN(eps=1.9, min_samples=3)
print (clf)
clf.fit(X)
labels = clf.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print ("\nClusters descubiertos: %d" % n_clusters_)

clustered = clf.labels_

ticker_count = len(returns.columns)
print ("Posibles pares en el universo de activos: %d " % (ticker_count*(
    ticker_count-1)/2))

clustered_series = pd.Series(index=returns.columns, data=clustered.flatten
    ())
clustered_series_all = pd.Series(index=returns.columns, data=clustered.
    flatten())
clustered_series = clustered_series[clustered_series != -1]

print(clustered_series_all.head(5))

CLUSTER_SIZE_LIMIT = 9999
counts = clustered_series.value_counts()
ticker_count_reduced = counts[(counts>1) & (counts<=CLUSTER_SIZE_LIMIT)]
print ("Clusters formados: %d" % len(ticker_count_reduced))
print ("Pairs para evaluar o validar: %d" % (ticker_count_reduced*(
    ticker_count_reduced-1)).sum())
```

La siguiente gráfica fue construida fue por medio de *t-distributed stochastic neighbor embedding* para reducir la dimensionalidad de los datos y permitir visualizar de manera gráfica los resultados del clustering de los activos en estudio, el siguiente cuadro de código permite aplicar el t-SNE.

```
X_tsne = TSNE(learning_rate=1000, perplexity=25, random_state=1337).
    fit_transform(X)
```

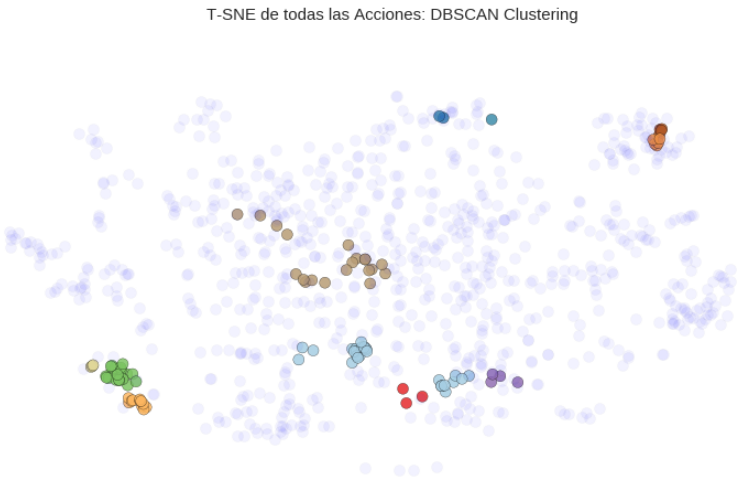
Se puede observar claramente la no inclusión de todos los activos dentro de los clusters, esto se da porque el algoritmo descarta aquellos activos que solo generan ruido en el cluster y no permiten llegar a la media óptima del cluster.

Los clusters no se construyen equitativamente con la muestra, por eso mismo es necesario identificar como es la composición de los clusters. El siguiente histograma muestra que efectivamente el número mínimo de activos por cluster es 3 y el máximo 27, así mismo que la mayoría de activos se concentran en los primeros 4 clusters.

Como se había mencionado previamente, se tuvo en cuenta también el criterio de cointegración. Esto con el fin de que los pares con los que se van a trabajar sean totalmente validos. Se aplicó el test de dos pasos propuesto por Engle-Granger (Engle et al., 1990) a todas las combinaciones posibles pares de activos presentes en los clusters. Para ello fue necesario tomar una función que iterara todas las posibles combinaciones de activos con el fin de evaluar la posible cointegración y dado un nivel de significacia del 5% se validará o no la potencial cointegración entre los pares de activos.

```
def find_cointegrated_pairs(data, significance=0.05):
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
```

FIGURA 3.1: DBSCAN Clustering

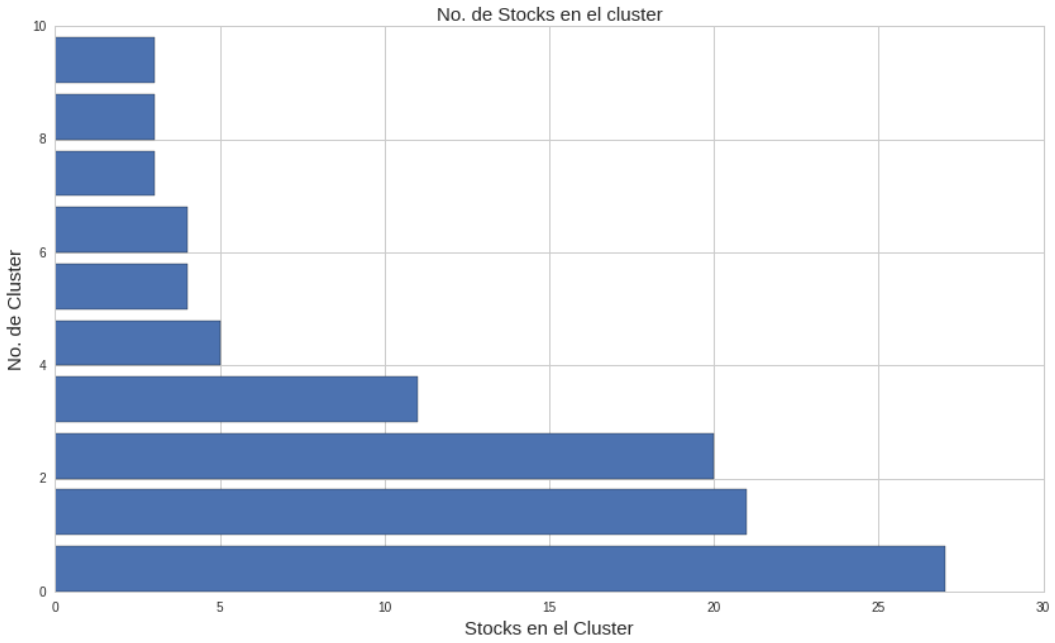



```
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < significance:
                pairs.append((keys[i], keys[j]))
    return(score_matrix, pvalue_matrix, pairs)

cluster_dict = {}
for i, which_clust in enumerate(ticker_count_reduced.index):
    tickers = clustered_series[clustered_series == which_clust].index
    score_matrix, pvalue_matrix, pairs = find_cointegrated_pairs(
        pricing[tickers])
    cluster_dict[which_clust] = {}
    cluster_dict[which_clust]['score_matrix'] = score_matrix
    cluster_dict[which_clust]['pvalue_matrix'] = pvalue_matrix
    cluster_dict[which_clust]['pairs'] = pairs
```

Al realizar el test a todos los posibles pares presentes en los clusters, se valida la existencia de 121 pares que cumplen ambos criterios con 72 activos no repetidos.

Debido a que se tiene una alta dimensionalidad de datos se repite el t-SNE con el fin de visualizar los pares validados. La siguiente gráfica muestra el relacionamiento de los activos y se puede evidenciar que este procedimiento fue efectivo ya que los activos solo se relacionan con otros activos que estén dentro de su mismo cluster. Así mismo es valido aclarar que la metodología planteada bajo los dos criterios no es redundante ya que se desecha un gran número de posibles pares.

FIGURA 3.2: Histograma Distribución de los Activos en los Clusters



3.3 Views: HMM

El objetivo de esta sección es explicar el procedimiento de construcción de uno de los input del modelo de asignación de activos, la matriz de views o expectativas $\boldsymbol{Q}$ del inversor. Serán views totalmente referenciales, es decir en función del rendimiento de otro activo.

En primera instancia es necesario estimar el HMM para todos los activos[3] por medio del algoritmo EM. Las variables de entrada para el modelo serán los spread de precios de cierre, apertura, máximo y mínimo $(S_t^c, S_t^o, S_t^{max}, S_t^{min})$ diarios de los pares de compañías validadas anteriormente dentro del periodo del 1 de Enero del 2008 hasta el 1 de Abril de 2018.
Se hará uso de la libreria `hmmlearn`, la cual que permite especificar parámetros iniciales y metodologías para estimar el HMM. Quantopian no tiene en su ecosistema instalada dicha librería, tiene la versión del HMM de `sklearn`, la cual no estima por el algoritmo EM el modelo y no permite ajustar otros parámetros necesarios. Por lo anterior, se hará la estimación de los views de manera local con datos de Bloomberg para el periodo de estudio, lo cual podría llegar a ser un riesgo operativo a la hora de integrar los resultados de diversas fuentes, sin embargo los resultados no se deberían ver afectados ya que una de las fuentes de datos de Quantopian es Bloomberg..

Para automatizar el acceso a la información se construye un ciclo que cree un archivo en Excel de tal forma que se aproveche el *Bloomberg Query Languaje* (BQL) para construir archivos con la información necesaria para la construcción de los views. Para ello, fue necesario construir las columnas con el nombre *Dates* y los campos nemotécnicos del último precio, precio de apertura, máximo y mínimo[4] dentro de la función `BFieldInfo("CAMPO NEMOTÉCNICO")`. El último paso para la construcción del archivo

[3] De ahora en adelante se entenderá como activo el spread entre los pares.
[4] `PX_LAST, PX_OPEN, PX_HIGH`,y `PX_LOW` respectivamente.

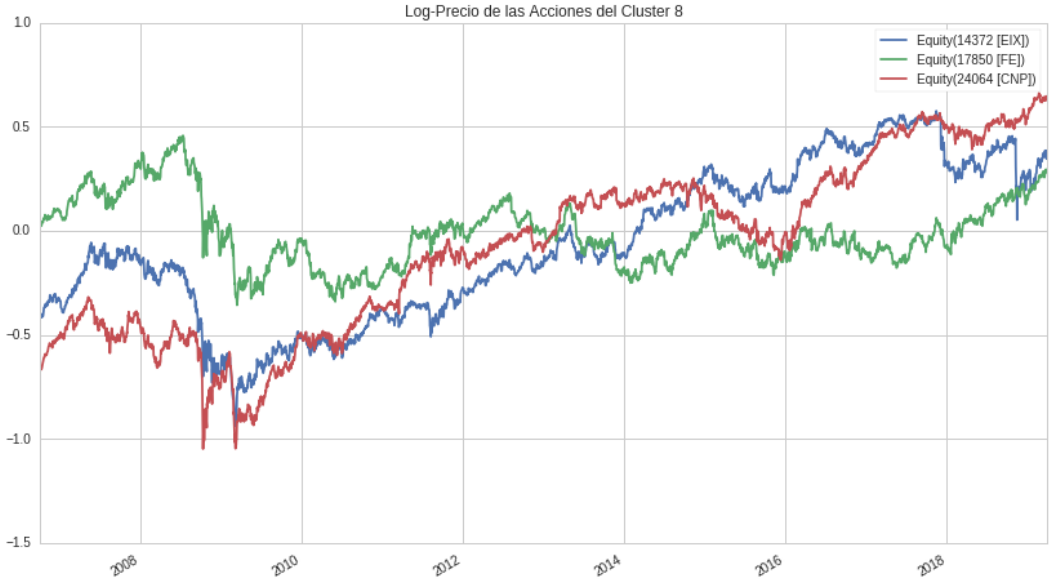
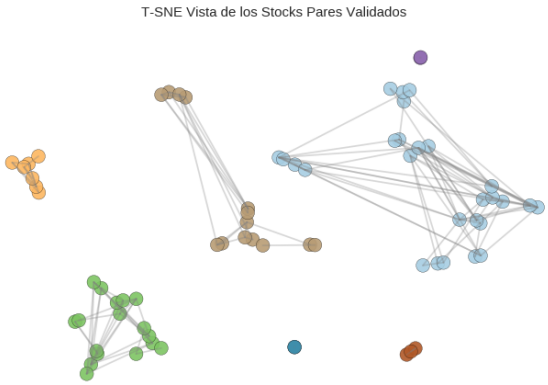


FIGURA 3.3: T-SNE Pares Validados por Cointegración



en Excel[5] que contendrá el query para la información será especificar el contenido de la celda `A2`, es decir, en la posición (1,0). El siguiente cuadro de código muestra la formula que debe contener esta celda para consultar el histórico de precios de la `ACCION_i` con frecuencia diaria desde el año 2000:

```
1 =BDH("ACCION_i US Equity";"PX_LAST,PX_OPEN,PX_HIGH,PX_LOW";"2000-01-01";""
      ;"Dir=V";"Fill=B";"Days=A";"Per=M";"Dts=S")
```

Los views estarán en función de los estados escondidos estimados por el HMM. Se tiene el set de datos $\{X_t(S_t^c, S_t^o, S_t^{max}, S_t^{min})\}$, el cual tiene m-estados dado que la cadena de Markov $C_t(m)$ tiene m-estados. Por lo tanto, la primer decisión que es necesario tomar, es identificar cual es número óptimo de estados escondidos. Este procedimiento se realizará de tal forma que el modelo con m-estados escondidos que minimiza los criterios de información de Akaike (Akaike, 1985) y Schwarz (Schwarz, 1978) e indicará cual es el número de estados óptimos m^*, esta metodología fue propuesta por (Nguyen, 2018). Se utilizan estos criterios de información ya que tienen en

[5]Para que los archivos se generen correctamente es necesario realizar este proceso en una terminal de Bloomberg con la cuenta abierta.

cuenta el score del modelo y premian (penalizan) la parsimonia (complejidad) del modelo estimado.

Teniendo el número de estados escondidos óptimos para la estimación del modelo m^* para cada uno de los diferentes activos (spread de los pares), se procederá a realizar la estimación del HMM para cada caso, con el objetivo de pronosticar el conjunto de precios para el siguiente periodo y con esto construir los views. Para realizar pronóstico se utilizará la matriz de transición de probabilidad denotada como $\mathbf{\Gamma}$ con el fin de predecir el siguiente estado y por consiguiente se calculará el valor esperado futuro haciendo uso de una simulación de montecarlo, calculando la diferencia del score del modelo dada la iteración anterior respecto a la actual para predecir el potencial cambio que existirá en los precios para el siguiente periodo según la metodología de (Tenyakov, 2014).

Para replicar la metodología explicada anteriormente es necesario iterar dentro de un rango finito de posibles estados (de 2 a 50) la estimación del modelo y por medio de los criterios de información, determinar cual es el número de estados óptimos para realizar pronosticos sobre los activos para la siguiente fecha. El siguiente cuadro de código permite identificar el número de estados óptimos:

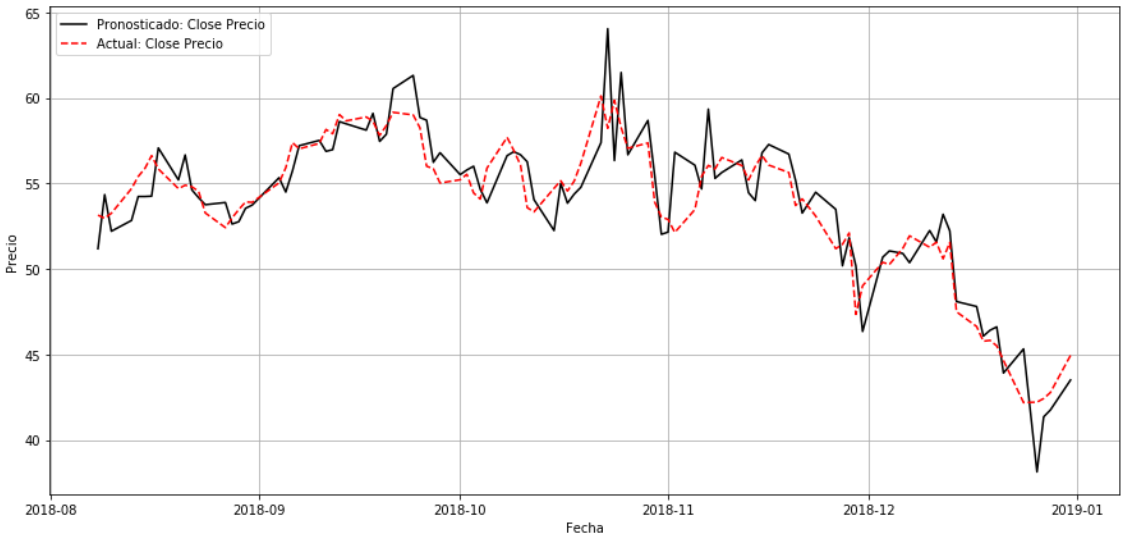
```
for stock in stocks:
    dataset = pd.read_excel('xxxxx.xlsx', sheet_name= stock)
    dataset = dataset.dropna()
    dataset.colums = ['Date','Close','Open','High','Low']
    dates = dataset.iloc[:,0]
    dataset = dataset.iloc[:,1:]
    dataset = dataset.values
    dataset = dataset[::-1]

    dates = dates[::-1]
    predicted_stock_data = np.empty([0,dataset.shape[1]])

    likelihood_vect = np.empty([0,1])
    aic_vect = np.empty([0,1])
    bic_vect = np.empty([0,1])

    for states in STATE_SPACE:

        num_params = states**2 + states
        model = hmm.GaussianHMM(n_components=states, covariance_type='full
    ', tol=0.0001, n_iter=NUM_ITERS)
        model.fit(dataset[NUM_TEST:,:])

        if model.monitor_.iter == NUM_ITERS:
            print('Aumentar el numero de iteraciones')
            sys.exit(1)

        likelihood_vect = np.vstack((likelihood_vect, model.score(dataset)
    ))

        aic_vect = np.vstack((aic_vect, -2 * model.score(dataset) + 2 *
    num_params))
        bic_vect = np.vstack((bic_vect, -2 * model.score(dataset) +
    num_params * np.log(dataset.shape[0])))

    opt_states = np.argmin(bic_vect) + 2

    print('El numero optimo de estados es {}'.format(opt_states)) # 45:13
```

Ahora es necesario estimar el modelo con el número de estados óptimos encontrados previamente. Por ello es necesario entrenar el modelo con aproximadamente el 80% de la muestra y generar un ciclo que me permita capturar los pronosticos y con ello tener un archivo organizado con los valores pronosticados para todas las fechas de interés y los activos estudiados. El siguiente cuadro de código permite replicar este proceso, el cual será vital para la generación de los views:

```
    for idx in reversed(range(NUM_TEST)):
        train_dataset = dataset[idx + 1:,:]
        test_data = dataset[idx,:];
        num_examples = train_dataset.shape[0]
        if idx == NUM_TEST - 1:
            model = hmm.GaussianHMM(n_components=opt_states,
    covariance_type='full', tol=0.0001, n_iter=NUM_ITERS, init_params='stmc
    ')
        else:
            model = hmm.GaussianHMM(n_components=opt_states,
    covariance_type='full', tol=0.0001, n_iter=NUM_ITERS, init_params='')
            model.transmat_ = transmat_retune_prior
            model.startprob_ = startprob_retune_prior
            model.means_ = means_retune_prior
            model.covars_ = covars_retune_prior
        model.fit(np.flipud(train_dataset))

        if model.monitor_.iter == NUM_ITERS:
            print('Aumentar el numero de iteraciones')
            sys.exit(1)
        iters = 1;
        past_likelihood = []
        curr_likelihood = model.score(np.flipud(train_dataset[0:K - 1, :])
    )
        while iters < num_examples / K - 1:
            past_likelihood = np.append(past_likelihood, model.score(np.
    flipud(train_dataset[iters:iters + K - 1, :])))
            iters = iters + 1

        likelihood_diff_idx = np.argmin(np.absolute(past_likelihood -
    curr_likelihood))
        predicted_change = train_dataset[likelihood_diff_idx,:] -
    train_dataset[likelihood_diff_idx + 1,:]
        predicted_stock_data = np.vstack((predicted_stock_data, dataset[
    idx + 1,:] + predicted_change))

    forecast = pd.DataFrame({'Fecha': dates[:NUM_TEST],'Close':dataset[
    range(NUM_TEST),0],
                             'Close_hat': np.flipud(predicted_stock_data
    [:,0]),
                             'Open':dataset[range(NUM_TEST),1],
                             'Open_hat': np.flipud(predicted_stock_data
    [:,1]),
                             'High':dataset[range(NUM_TEST),2],
                             'High_hat': np.flipud(predicted_stock_data
    [:,2]),
                             'Low':dataset[range(NUM_TEST),3],
                             'Low_hat': np.flipud(predicted_stock_data[:,3])
    })

    mape = calc_mape(predicted_stock_data, np.flipud(dataset[range(
    NUM_TEST),:]))

    print('MAPE para el stock {} es '.format(stock),mape)
```

Para ilustrar la capacidad predictiva de esta metodología de pronostico, la siguiente figura permite observar el ajuste del pronostico del spread *versus* los precios de cierre observados para una ventana de tiempo seguida y aleatoria de 100 días, la cual es óptima teniendo en cuenta las iteraciones de posibles ventanas que aconseja realizar (Zhang, 2004) para potencializar la capacidad predictiva del modelo. La ventana fue desde Agosto de 2018 hasta Enero de 2019.

FIGURA 3.4: Spread `ATO US Equity - SCG US Equity`



Para tener una medida de incertidumbre media sobre los pronósticos se calculará el error porcentual absoluto medio, que esta dado por la siguiente fórmula:

$$\text{MAPE} = \frac{1}{n}\sum_{t=t_0}^{T}\left|\frac{S_t - \hat{S}_t}{S_t}\right|$$

Por ejemplo, para el caso expuesto en la Figura anterior (`ATO US Equity - SCG US Equity`) el MAPE es de 2.37%. Lo cual permite tener una banda de precios alrededor de los pronosticados para hacer una posible construcción de los views, y su incertidumbre.

Ahora, se utilizarán los pronósticos como views sobre los spreads. Con el fin de evitar sobredimensionar la optimización se construirán views referenciales, es decir, el spread 1 tendrá mayor, menor o igual rendimiento respecto al spread 2; se organizarán los views de pares de spreads aleatorios en archivos históricos con el fin de balancear el portafolio óptimamente día a día. Por último es necesario organizar los views de tal forma que sea posible construir las matrices $\boldsymbol{P}$ y $\boldsymbol{Q}$ para implementarlas en el modelo de Black-Litterman, la siguiente función permite realizar el proceso deseado mientras los views esten dentro de una lista con formato de dictado indicando la relación (mayor, menor o igual) entre un activo y el de referencia:

```
def views_mtx(tickers, views):
    r, c = len(views), len(tickers)
    Q = [views[i][3] for i in range(r)]
    P = np.zeros([r, c])
    tickersToIndex = dict()
    for i, n in enumerate(tickers):
```

```
        tickersToIndex[n] = i
    for i, v in enumerate(views):
        name1, name2 = views[i][0], views[i][2]
        P[i, tickersToIndex[name1]] = +1 if views[i][1] == '>' else -1
        P[i, tickersToIndex[name2]] = -1 if views[i][1] == '>' else +1
    return np.array(Q), P
```

3.4 Black-Litterman

Se cuenta con todos los inputs necesarios para aplicar el modelo de Black-Litterman de forma iterativa durante todos los días del periodo de estudio, capitalización de mercado histórica, views referenciales históricos y precios históricos de los spreads.

Por lo tanto lo único que haría falta es determinar la tasa libre de riesgo r_f y el factor de escalamiento τ. Para ello, se descargarán las tasas de los bonos de EEUU con plazo de 10 años para tener una proxy de la tasa libre de riesgo y el factor τ será 0.25 para evitar subestimar el factor (Meucci, 2010).

Cabe recordar que la estrategia se implementará desde el 1 de Diciembre de 2018 hasta el 1 de Abril de 2019, por lo tanto se estimarán los precios óptimos iterativamente para cada uno de los días que comprenden el periodo de interés. Así mismo es necesario tener claro el hecho que se esta utilizando una ventana de 500 días, para evitar sesgar el vector de retornos esperados y la matriz de covarianza. En el capítulo Apéndices, en la primera sección, se encuentran las funciones necesarias para implementar el modelo y obtener los pesos óptimos para cada fecha.

3.5 Estrategia de Negociación: Pairs Trading

Al tener los pesos óptimos calculados por medio del modelo de Black-Litterman para cada activo (spread entre los pares), estos se utilizarán de manera prescriptiva más no absoluta. Es decir, como una recomendación de límites de inversión para los activos; es necesario tener en cuenta como es la estructura natural del activo:

$$A_{t,[a,b]} = S_{t,[a]} - S_{t,[b]}$$

Luego, si se esta comprando el activo $A_{t,[a,b]}$ es lo mismo que comprar una unidad de $S_{t,[a]}$ y vender una de $S_{t,[b]}$ y cuando se dé el caso de venta sobre $A_{t,[a,b]}$ se vendería $S_{t,[a]}$ y compraría $S_{t,[b]}$.

La estrategia de trading de alta frecuencia se basará como antes se propuso, en pairs trading. Se estimará el estado escondido con el fin de identificar cambios de tendencia y junto a medidas que permiten encontrar valores "extremos" como son los z-scores (z) se emitirán ordenes de compra y venta con frecuencia de minutos acogiéndose a los límites en los pesos del portafolio total encontrados por medio de Black-Litterman.

Para efectuar la estrategia de manera eficiente es necesario interpretar los estados escondidos para evitar generar ordenes en el mercado que sean incongruentes.

$$\text{order}(A_{t,[a,b]})_t = \begin{cases} \hat{C}_t^d \quad \text{y} \quad z^u \Rightarrow -1 \times w^*_{A_{t,[a,b]}} \\ \hat{C}_t^m \quad \text{y} \quad z^i \Rightarrow 0 \\ \hat{C}_t^u \quad \text{y} \quad z^d \Rightarrow 1 \times w^*_{A_{t,[a,b]}} \end{cases} \tag{3.1}$$

Donde z^u es el z-score superior (1 porque es a una desviación estándar de la media que es 0 porque esta estandarizado el proceso) y z^d el inferior. De la misma forma $\hat{C}_t^d$ es el estado escondido estimado que indica que existira cambio de tendencia a la baja y $\hat{C}_t^u$ a la alza. Para las ordenes se maneja 1 para comprar y -1 para vender, multiplicado por el peso asignado para cada activo por el modelo de Black-Litterman.

3.6 Ensamblaje y Backtesting

Es necesario tener en cuenta que se esta trabajando con dos frecuencias de datos, diarios y minutos. Se trabaja así para alimentar el modelo de Black-Litterman con precios diarios, gracias a que el rebalanceo del portafolio se realiza diariamente (excepto si las operaciones abiertas tienen ganancias acumuladas diarias que respeten los límites recomendados por el modelo, tienen menos de cinco días de abiertas y no sufren una desvalorización de sus retornos diarios superior al ocho porciento), mientras que se hace uso de datos intra-día para ejecutar la estrategia de trading sujeto a los límites diarios del modelo de portafolio. En resumen, el horizonte de inversión es diario, pero la estrategia se ejecuta en minutos.

Para realizar el backtesting del algoritmo fue necesario acceder al ecosistema de algoritmos en la plataforma de Quantopian, la diferencia con el de investigación es poca, pero si es necesario manejar otras librerías y manejo lógico de las estructuras. Las librerias necesarias para trabajar son las siguientes:

```
import numpy as np
import pandas as pd
import quantopian.optimize as opt
import quantopian.algorithm as algo
from sklearn.hmm import GaussianHMM
```

Numpy y Pandas para el manejo de datos y funciones matemáticas, las librerías nativas del ecosistema de algoritmos Algorithm y Optimize para formar el algoritmo y poder realizar las optimizaciones del portafolio en tiempo reducido, por último la sección descontinuada de HMM en Sklearn para poder estimar e interpretar los estados escondidos de los activos.

Después de importar las librerías es necesario definir los parámetros iniciales del algoritmo, los cuales serán los tickers de los activos, la ventana de tiempo para calcular los z-scores y las indicaciones de operación corta y larga. Así mismo es necesario indicar los horarios de activación del algoritmo y ejecución de las reglas. Para inicializar el algoritmo en este ambiente se realiza definiendo funciones como la siguiente:

```
def initialize(context):
    schedule_function(handle_data, date_rules.every_day(), time_rules.
    market_close(minutes=60))
    set_commission(commission.PerTrade(cost=0.01))
    context.expo = ExposureMngr(target_leverage = 4.0,
    target_long_exposure_perc = 0.50, target_short_exposure_perc = 0.50)
    context.holding_minutos = 7200
```

```
    context.stock1 = symbol('BMY')
                .
                .
    context.stockn = symbol('GIS')
    context.stocks = [(context.stock1, context.stock2),...,(context.
    stocknn, context.stockn)]
    context.corto = 1
    context.largo = -1
    context.horizonte = opt.g_day #174240
    context.hmm_window = 100
    context.dia = 1
    context.corto_spread = False
    context.largo_spread = False
    context.num_pairs = len(context.stocks)
```

Después de realizar una optimización y balanceo del portafolio por medio de numerosos backtestings se encontró que la ventana de tiempo óptima para que el modelo tenga de referencia son 121 días, es decir 174240 minutos. Así mismo fue necesario notificar que el algoritmo que su funcionamiento es hasta 1 hora antes que el mercado cierre para evitar estar expuestos en cierta volatilidad de este periodo, indicar el número máximo de minutos que se puede mantener abierta una posición (5 días,es decir, 7200 minutos), también especificar el costo por operación de 1% y por último declarar el máximo grado de exposición[6] (será de 4 veces su valor de mercado) del portafolio distribuido 50% para las posiciones cortas y 50% para las largas.

Posteriormente es necesario construir la columna del algoritmo, donde se encontrará el llamado y manipulación de los datos, y las reglas para operar. Para inicializar esta sección del algoritmo es necesario definir la función `handle_data(context, data)`, la cual se alimenta de los parámetros definidos como `context` previamente. Ya que se quiere realizar trading de alta frecuencia se hará llamada a los datos minuto a minuto, si se quisiera llamar los precios de los últimos 174240 minutos del primer par de activos se realizaría el siguiente proceso:

```
def handle_data(context, data):
    s1 = context.stock1
    s2 = context.stock2
    prices = data.history([s1, s2], "price", context.horizonte, '1m')
```

Es posible llamar los precios de cierre, apertura, mínimo, máximo y volumen para lapsos de tiempo de un minuto. Así mismo se puede cambiar la frecuencia y ventana de tiempo, por ejemplo, para identificar los estados escondidos del spread se necesita correr el modelo con precios diarios para clasificar los intradía, para ello se ejecuta la siguiente sección dentro de la función `handle_data(context, data)`:

```
def handle_data(context, data):
            .
            .
            .
    prices_ = data.history([s1, s2], "price", 121, '1d')
    p_open = data.history([s1, s2], "open", 121, '1d')
    p_max = data.history([s1, s2], "high", 121, '1d')
    p_min = data.history([s1, s2], "low", 121, '1d')
    spread_real = prices_[s1] - prices_[s2]
    spread_open = p_open[s1] - p_open[s2]
    spread_max = p_max[s1] - p_max[s2]
    spread_min = p_min[s1] - p_min[s2]
```

[6] Ver capítulo Apéndices, sección Código de Función de Exposición

Con estos insumos ya es posible calcular el z-score minuto a minuto y entrenar el modelo con los datos diarios pero estimar con los datos por minuto. Para evitar restricciones de memoria se estiman los modelos con 10 iteraciones y los datos se dejan en el formato específico que exige la libreria para trabajar el modelo. El siguiente cuadro de código permite replicar el proceso previamente mencionado para el mismo par de activos con los que se viene ejemplificando el trabajo del algoritmo:

```
def handle_data(context, data):
            .
            .
            .
    mean_ = np.mean(prices[s1] - prices[s2])
    std_ = np.std(prices[s1] - prices[s2])
    prices_ventana = prices.iloc[-context.dia:]
    spread = prices_ventana[s1] - prices_ventana[s2]

    X = np.column_stack([spread_real.values, spread_open.values,
    spread_max.values, spread_min.values])
    X = pd.DataFrame(X)

    for k in X.columns:
        X[k] = X[k].fillna(X[k].mean())

    X = X.values

    model = GaussianHMM(context.opt_hmm_states, covariance_type="diag",
    n_iter=10, means_weight=0.5)
    model.fit([X])
    hmm = pd.DataFrame({'precio':spread_real.values,'hmm':model.predict(X)
    })
    hmm_mean = hmm.groupby('hmm').mean().sort_values('precio',ascending=
    True)

    estados_up_dw = [hmm_mean.index.tolist()[-1], hmm_mean.index.tolist()
    [0]]

    X_ = np.column_stack([(prices[s1]-prices[s2]).values[-1], (prices1[s1
    ]-prices1[s2]).values[-1], (prices2[s1]-prices2[s2]).values[-1], (
    prices3[s1]-prices3[s2]).values[-1]])
```

El último paso es aplicar la estrategia propuesta en la ecuación (2.1), conectando las ordenes de compra con los pesos propuestos por el modelo de Black-Litterman para el día en que se esta operando. La interpretación de los estados se realiza bajo un criterio de organización por valores extremos, entonces al predecir el estado escondido dados los datos en el minuto t, si el estado escondido resultante en t es alguno de los dos estados considerados valores extremos se abre la operación dependiendo de la ecuación (2.1).

Es necesario tener en cuenta que los pesos calculados en Black-Litterman son prescriptivos, por eso al generar la orden se permite un margen de error, que en este caso es de 2%. Otra aclaración necesaria es que se instauro como política general del algoritmo cerrar posiciones que estén asociadas a una perdida de 8%, es decir, tener un *stop loss* equivalente a 8% independientemente del activo. El 2% de tolerancia al error de los pesos y el 8% de *stop loss* se establecen asumiendo que el inversionista tiene un perfil moderado de riesgo, estos criterios podrían ser modificados para otro perfil de inversionista. Con el fin de ejemplificar un corto sobre el spread de los dos activos con los que se ha mostrado el funcionamiento del algoritmo, la sección del código que abriría el corto seria la siguiente:

```
def handle_data(context, data):

        s1_target_shares = -1
        s2_target_shares = 1
        context.corto_spread[i] = True
        context.largo_spread[i] = False

        (s1_target_pct, s2_target_pct) = computeHoldingsPct(
    s1_target_shares, s2_target_shares, s1[-1], s2[-1] )
        context.target_weights[s1] = s1_target_pct * (2.0) * w_opt[p][q]
        context.target_weights[s2] = s2_target_pct * (2.0) * w_opt[p][q+1]
        record(s1_pct=s1_target_pct, s2_pct=s2_target_pct)
        allocate(context, data)
        return
```

Para realizar el backtesting es necesario aprovechar la opción de Quantopian llamada *Build Algorithm*, la cual permite comprobar la viabilidad de su ejecución y posteriormente realizar un *Run Full Backtest*, el cual permitirá comparar el rendimiento del algoritmo contra el del indice *S&P500* y obtener algunas metricas de riesgo y desempeño. Quantopian trata de emular la realidad de tal forma que van a existir ordenes que no sean efectivas en el mercado ya sea por restricciones de liquidez, retraso en los precios de mercado y otros factores del mercado.

Para ilustrar los grandes requerimientos de procesamiento del algoritmo construido, el backtesting de un par activos dura aproximadamente 7 horas para el periodo de Diciembre de 2018 hasta Abril del 2019 dado que se quieren obtener todas las métricas y gráficos interactivos. Se realizo el backtesting del algoritmo como si el portafolio estuviera únicamente conformado por `ATO US Equity - SCG US Equity`, lo cual arrojo retornos de 360.62% en cuatro meses, con un apalancamiento máximo de 2 veces para cortos y 2 para largos. La siguiente captura de pantalla contiene los resultados del backtesting completo del par de activos mencionados anteriormente:

FIGURA 3.5: Backtesting `ATO US Equity - SCG US Equity`

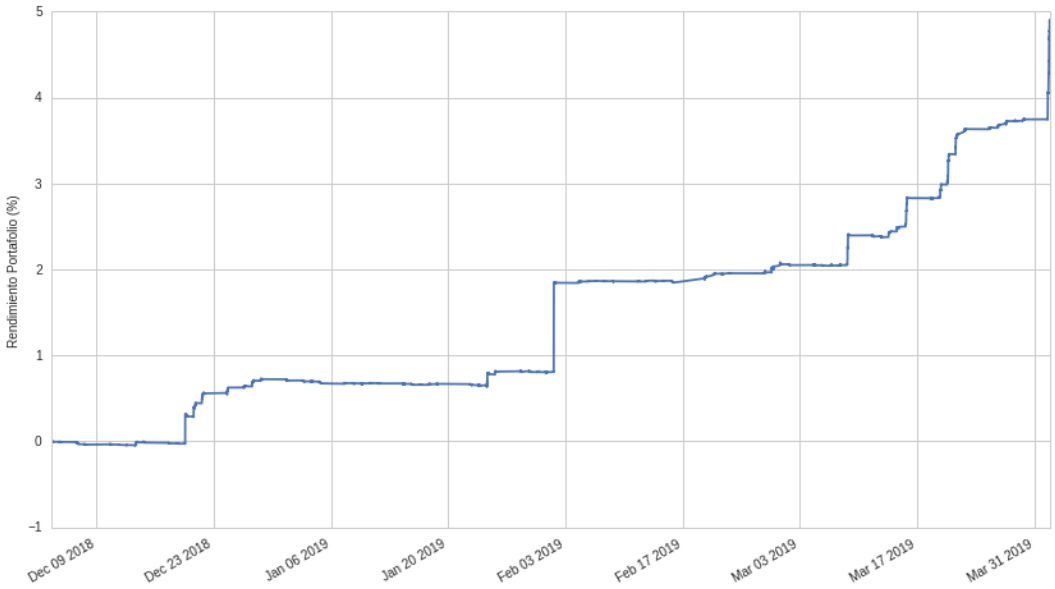
Para realizar el backtesting de los 121 pares encontrados en la selección de activos fue necesario construir el algoritmo en el ambiente de algoritmos, utilizar la opción de *Build Algorithm* para testear la viabilidad del algoritmo, lo cual tuvo una duración aproximada de 27 horas. Para optimizar el proceso de backtesting, o realizarlo de

una forma que requiriera menor tiempo, se generó un código de algoritmo para realizar la evaluación del algoritmo en el ecosistema de investigación, ya que tiene mayor capacidad de procesamiento. Para realizar este proceso se requiere correr el siguiente código, teniendo el ID del algoritmo y estando en línea en la plataforma de Quantopian:

```
algo_id = 'xxxxxxxxxxxxxxxxxxxx'
result = get_backtest(algo_id)
result.create_full_tear_sheet()
result.preview('risk')
```

Gracias a este método, el backtesting tarda 16 horas. Los datos resultantes del backtesting permiten acceder al histórico del valor de mercado del portafolio, por lo tanto al retorno histórico del portafolio administrado por el algoritmo. Organizando los datos se construye la gráfica de los retornos del portafolio (Figura 2.6), la cual permite observar el excelente desempeño del algoritmo para este periodo de tiempo, se logran materializar 487% de retornos *versus* 0.12% del indice `S&P500`.

FIGURA 3.6: Backtesting Final del Algoritmo - Rendimiento del Portafolio



Capítulo 4

Conclusiones

La existencia de plataformas como Quantopian facilitan la investigación y desarrollo en temas relacionados con trading algorítmico. Este trabajo aprovecha gran parte de las funcionalidades de esta plataforma, sin embargo, al utilizar recursos externos a la plataforma, como fueron los datos de Bloomberg y los archivos de views históricos, el algoritmo se expone a un riesgo operativo externo. Si bien es claro que la integración de los datos externos a Quantopian es posible, fue necesario realizar este proceso con mucho detalle y precaución, el menor cambio en la integración de los views podría haber generado un cambio considerable en la composición de los límites de los activos del portafolio.

Una de las ventajas que ofrece este algoritmo es la posibilidad de calibración según el perfil de riesgo del inversionista que este dispuesto a utilizarlo. Por un lado se puede calibrar el apalancamiento y el *stop loss*, y por el otro se puede aumentar o disminuir el número de estados escondidos estimados por el HMM, con el fin de volver más o menos flexible el criterio de decisión del algoritmo ante los valores extremos.

El rendimiento del algoritmo es destacable, dado que supera en rendimiento al índice de referencia del mercado por mas de 485% en un periodo de tiempo relativamente reducido. Así mismo, se tiene una exposición moderada, un *drawdown* totalmente controlado por el *stop loss* y el rebalanceo del portafolio periódicamente. El algoritmo tiene muy buen desempeño en términos de retornos, sin embargo podría mejorar su efectividad en la ejecución de operaciones, ya que para el caso del par de activos con los que se ejemplifico en el pasado capitulo, el máximo[1] retorno posible era 814.85% para el periodo de estudio y se alcanzo solamente 360.62%.

Es claro que la exposición al mercado de Estados Unidos es alta por la composición del portafolio (únicamente en activos del mercado de E.E.U.U), sin embargo, esta se podría mitigar ampliando la selección de activos ingresando datos de otros mercados[2]. La estrategia implementada para minimizar el riesgo para este trabajo fue no enfocarse en industrias particulares, sino aprovechar todo el universo de activos y por lo tanto diversificar la composición del portafolio para evitar concentración en sectores específicos y además aprovechar modelos como el de Black-Litterman como guía de donde ubicar el capital de forma óptima dadas unas perspectivas calculadas con el pronostico del HMM.

[1] Dado que se hubieran aprovechado todas las oportunidades de inversión perfectamente.

[2] Este proceso sería externo, ya que Quantopian tiene datos únicamente del mercado estadounidense.

4.1 Trabajo Futuro

Actualmente se esta trabajando en como mejorar la estimación de la matriz de probabilidades de transición o proponer una nueva estimación de esta matriz $\mathbf{\Gamma}$ de tal forma que la probabilidad contemple la secuencia de estados escondidos pasados y se introduzca un marco de agotamiento que indique cuando es inminente un cambio de estado dado que la secuencia de ese mismo estado esta agotada o llego al máximo. El objetivo sería tener una alternativa de matriz de transición de probabilidad relajando algunos supuestos, pero incorporandola al modelo original.Sus componentes i, j estarían representados por la siguiente ecuación:

$$\gamma_{ij}(t) = \mathbb{P}(C_{t+s}|\{C\}_{t=\tau-s-1}^{s} = i \wedge \max\{\{C\}_{t=\tau^*-s-1}^{s} = \bar{i}\})$$

donde se tiene la secuencia de estados desde $\tau - s - 1$ hasta $t + s$ y el máximo número de veces que la secuencia ha repetido el estado $\bar{i}$ para una ventana de tiempo determinada.

La metodología propuesta anteriormente podría llegar a modificar algunos de los resultados, para evitar asumir que la probabilidad de cambio de estados no dependa de la secuencia histórica, lo cual permitiría mejorar el criterio de decisión de inversión por parte de la estrategia de este trabajo.

Por otro lado, se podría mejorar el algoritmo implementando modelos de riesgo que regulen y determinen la toma de decisiones de inversión. Así mismo, sería óptimo la inclusión de mercados extranjeros para evitar estar sobre-expuesto a un solo país. De igual forma, encontrar el apalancamiento óptimo dados unos criterios de riesgo podría llegar a mejorar el rendimiento del algoritmo.

Teniendo en cuenta la alta capacidad predictiva del modelo HMM, se esta trabajando en la estimación de bandas de precios para implementar estrategias con opciones de venta de volatilidad, lo cual podría llevar al algoritmo a empezar a trabajar con derivados.

Capítulo 5

Apendice

5.1 Código Black-Litterman

```
def port_mean(W, R):
    return sum(R * W)
def port_var(W, C):
    return np.dot(np.dot(W, C), W)
def port_mean_var(W, R, C):
    return port_mean(W, R), port_var(W, C)
def solve_frontier(R, C, rf):
    def fitness(W, R, C, r):
        mean, var = port_mean_var(W, R, C)
        penalty = 100 * abs(
            mean - r)
        return var + penalty
    frontier_mean, frontier_var = [], []
    n = len(R)
    for r in np.linspace(min(R), max(R), num=20):
        W = np.ones([n]) / n
        b_ = [(0, 1) for i in range(n)]
        c_ = ({'type': 'eq', 'fun': lambda W: sum(W) - 1.})
        optimized = scipy.optimize.minimize(fitness, W, (R, C, r), method=
    'SLSQP', constraints=c_, bounds=b_)
        frontier_mean.append(r)
        frontier_var.append(port_var(optimized.x, C))
    return np.array(frontier_mean), np.array(frontier_var)
def solve_weights(R, C, rf):
    def fitness(W, R, C, rf):
        mean, var = port_mean_var(W, R, C)
        util = (mean - rf) / np.sqrt(var)
        return 1 / util
    n = len(R)
    W = np.ones([n]) / n
    b_ = [(0., 1.) for i in range(n)]
    c_ = ({'type': 'eq', 'fun': lambda W: sum(W) - 1.})
    optimized = scipy.optimize.minimize(fitness, W, (R, C, rf), method='
    SLSQP', constraints=c_, bounds=b_)
    if not optimized.success: print('Ojo con los pesos')
    return optimized.x
class Result:
    def __init__(self, W, tan_mean, tan_var, front_mean, front_var):
        self.W=W
        self.tan_mean=tan_mean
        self.tan_var=tan_var
        self.front_mean=front_mean
        self.front_var=front_var
    def optimize_frontier(R, C, rf):
        W = solve_weights(R, C, rf)
        tan_mean, tan_var = port_mean_var(W, R, C)
```

```
45          front_mean, front_var = solve_frontier(R, C, rf)
46          return Result(W, tan_mean, tan_var, front_mean, front_var)
47      def display_assets(names, R, C, color='black'):
48          plt.scatter([C[i, i] ** .5 for i in range(n)], R, marker='x',
    color=color)
49          for i in range(n):
50              plt.text(C[i, i] ** .5, R[i], '  %s' % names[i],
    verticalalignment='center', color=color)
51      def display_frontier(result, label=None, color='black'):
52          plt.text(result.tan_var ** .5, result.tan_mean, '    tangent',
    verticalalignment='center', color=color)
53          plt.scatter(result.tan_var ** .5, result.tan_mean, marker='o',
    color=color)
54          plt.plot(result.front_var ** .5, result.front_mean, label=label,
    color=color)
55  def load_data():
56      syymbols = sym_finales
57      cap = dictado_final
58      n = len(syymbols)
59      prices_out, caps_out = [], []
60      for s in syymbols:
61          dataframe001 = get_pricing(
62              symbols=[tickers['a'].tolist()[tickers['sym'].tolist().index(s
    )],tickers['b'].tolist()[tickers['sym'].tolist().index(s)]],
63              fields='close_price',
64              start_date=pd.Timestamp(study_date) - pd.DateOffset(months=25)
    ,
65              end_date=pd.Timestamp(study_date))
66          dataframe = pd.DataFrame(dataframe001)
67          dataframe['close'] = dataframe.iloc[:,0] - dataframe.iloc[:,1]
68          prices = list(dataframe['close'])[-500:]
69          prices_out.append(prices)
70          caps_out.append(cap[s])
71      return symbols, prices_out, caps_out
72  namees, prices, caps = load_data()
73  n = len(sym_finales)
74  namees = sym_finales
75  def assets_historical_returns_and_covariances(prices):
76      prices = np.matrix(prices)
77      rows, cols = prices.shape
78      returns = np.empty([rows, cols - 1])
79      for r in range(rows):
80          for c in range(cols - 1):
81              p0, p1 = prices[r, c], prices[r, c + 1]
82              returns[r, c] = (p1 / p0) - 1
83      expreturns = []
84      for r in range(rows):
85          expreturns.append(np.mean(returns[r]))
86      covars = np.cov(returns)
87      expreturns = np.array(expreturns)
88      expreturns = (1 + expreturns) ** 252 - 1
89      covars = covars * 252
90      return expreturns, covars
91  W = np.array(caps) / sum(caps)
92  R, C = assets_historical_returns_and_covariances(prices)
93  rf = .0425
94  display(pd.DataFrame({'Return': R, 'Weight (based on market cap)': W},
        index=namees).T)
95  res1 = optimize_frontier(R, C, rf)
96  display(pd.DataFrame({'Weight_0': W,'Weight_opt': res1.W}, index=namees).T
        )
97  mean, var = port_mean_var(W, R, C)
98  lmb = (mean - rf) / var
```

```
Pi = np.dot(np.dot(lmb, C), W) o
res2 = optimize_frontier(Pi+rf, C, rf)
display(pd.DataFrame({'Weight_0': W,'Weight_opt_0': res1.W,'Weight_opt_1':
     res2.W}, index=namees).T)
def views_mtx(tickers, views):
    r, c = len(views), len(tickers)
    Q = [views[i][3] for i in range(r)]
    P = np.zeros([r, c])
    tickersToIndex = dict()
    for i, n in enumerate(tickers):
        tickersToIndex[n] = i
    for i, v in enumerate(views):
        name1, name2 = views[i][0], views[i][2]
        P[i, tickersToIndex[name1]] = +1 if views[i][1] == '>' else -1
        P[i, tickersToIndex[name2]] = -1 if views[i][1] == '>' else +1
    return np.array(Q), P
views = pd.read_excel('views_hist_tesis.xlsx',sheetname=fecha_int).tolist
    ()
Q, P = create_views_and_link_matrix(namees, views)
print('Views Matrix')
display(pd.DataFrame({'Views':Q}))
print('Link Matrix')
display(pd.DataFrame(P))
tau = .025
omega = np.dot(np.dot(np.dot(tau, P), C), np.transpose(P))
sub_a = np.linalg.inv(np.dot(tau, C))
sub_b = np.dot(np.dot(np.transpose(P), np.linalg.inv(omega)), P)
sub_c = np.dot(np.linalg.inv(np.dot(tau, C)), Pi)
sub_d = np.dot(np.dot(np.transpose(P), np.linalg.inv(omega)), Q)
Pi_adj = np.dot(np.linalg.inv(sub_a + sub_b), (sub_c + sub_d))
res3 = optimize_frontier(Pi_adj + rf, C, rf)

display(pd.DataFrame({'Weight': res2.W,'Weight_Adj': res3.W}, index=namees
    ).T)
```

5.2 Código de Función de Exposición

```
class ExposureMngr(object):
    def __init__(self, target_leverage = 4.0, target_long_exposure_perc =
    0.50, target_short_exposure_perc = 0.50):
        self.target_leverage            = target_leverage
        self.target_long_exposure_perc  = target_long_exposure_perc
        self.target_short_exposure_perc = target_short_exposure_perc
        self.short_exposure             = 0.0
        self.long_exposure              = 0.0
        self.open_order_short_exposure  = 0.0
        self.open_order_long_exposure   = 0.0
    def get_current_leverage(self, context, consider_open_orders = True):
        curr_cash = context.portfolio.cash - (self.short_exposure * 2)
        if consider_open_orders:
            curr_cash -= self.open_order_short_exposure
            curr_cash -= self.open_order_long_exposure
        curr_leverage = (context.portfolio.portfolio_value - curr_cash) /
    context.portfolio.portfolio_value
        return curr_leverage
    def get_exposure(self, context, consider_open_orders = True):
        long_exposure, short_exposure = self.get_long_short_exposure(
    context, consider_open_orders)
        return long_exposure + short_exposure
    def get_long_short_exposure(self, context, consider_open_orders = True
    ):
        long_exposure          = self.long_exposure
```

```
22          short_exposure        = self.short_exposure
23          if consider_open_orders:
24              long_exposure  += self.open_order_long_exposure
25              short_exposure += self.open_order_short_exposure
26          return (long_exposure, short_exposure)
27      def get_long_short_exposure_pct(self, context, consider_open_orders =
    True, consider_unused_cash = True):
28          long_exposure, short_exposure = self.get_long_short_exposure(
    context, consider_open_orders)
29          total_cash = long_exposure + short_exposure
30          if consider_unused_cash:
31              total_cash += self.get_available_cash(context,
    consider_open_orders)
32          long_exposure_pct   = long_exposure  / total_cash if total_cash >
    0 else 0
33          short_exposure_pct  = short_exposure / total_cash if total_cash >
    0 else 0
34          return (long_exposure_pct, short_exposure_pct)
35      def get_available_cash(self, context, consider_open_orders = True):
36          curr_cash = context.portfolio.cash - (self.short_exposure * 2)
37          if consider_open_orders:
38              curr_cash -= self.open_order_short_exposure
39              curr_cash -= self.open_order_long_exposure
40          leverage_cash = context.portfolio.portfolio_value * (self.
    target_leverage - 1.0)
41          return curr_cash + leverage_cash
42      def get_available_cash_long_short(self, context, consider_open_orders
    = True):
43          total_available_cash  = self.get_available_cash(context,
    consider_open_orders)
44          long_exposure         = self.long_exposure
45          short_exposure        = self.short_exposure
46          if consider_open_orders:
47              long_exposure  += self.open_order_long_exposure
48              short_exposure += self.open_order_short_exposure
49          current_exposure       = long_exposure + short_exposure +
    total_available_cash
50          target_long_exposure  = current_exposure * self.
    target_long_exposure_perc
51          target_short_exposure = current_exposure * self.
    target_short_exposure_perc
52          long_available_cash   = target_long_exposure  - long_exposure
53          short_available_cash  = target_short_exposure - short_exposure
54          return (long_available_cash, short_available_cash)
55      def update(self, context, data):
56          self.open_order_short_exposure  = 0.0
57          self.open_order_long_exposure   = 0.0
58          for stock, orders in  get_open_orders().iteritems():
59              price = data.current(stock, 'price')
60              if np.isnan(price):
61                  continue
62              amount = 0 if stock not in context.portfolio.positions else
    context.portfolio.positions[stock].amount
63              for oo in orders:
64                  order_amount = oo.amount - oo.filled
65                  if order_amount < 0 and amount <= 0:
66                      self.open_order_short_exposure += (price * -
    order_amount)
67                  elif order_amount > 0 and amount >= 0:
68                      self.open_order_long_exposure  += (price *
    order_amount)
69          self.short_exposure = 0.0
70          self.long_exposure  = 0.0
```

```
71          for stock, position in context.portfolio.positions.iteritems():
72              amount = position.amount
73              last_sale_price = position.last_sale_price
74              if amount < 0:
75                  self.short_exposure += (last_sale_price * -amount)
76              elif amount > 0:
77                  self.long_exposure  += (last_sale_price * amount)
```

Bibliografía

Akaike, Hirotugu (1985). "Prediction and entropy". In: *Selected Papers of Hirotugu Akaike*. Springer, pp. 387–410.

Black, Fischer and Robert Litterman (1990). *Asset allocation: combining investor views with market equilibrium*. Tech. rep. Discussion paper, Goldman, Sachs & Co.

Engle, Robert F et al. (1990). "Seasonal integration and cointegration". In: *Journal of econometrics* 44.1-2, pp. 215–238.

Feller, William (1957). "An introduction to probability theory and its applications". In: *Taylor & Francis*.

Gatev, Evan, William N Goetzmann, and K Geert Rouwenhorst (2006). "Pairs trading: Performance of a relative-value arbitrage rule". In: *The Review of Financial Studies* 19.3, pp. 797–827.

Grinold, Richard C and Ronald N Kahn (2000). "Active portfolio management". In:

Harris, Richard and Robert Sollis (2003). "Applied time series modelling and forecasting". In:

He, Guangliang and Robert Litterman (1999). "The intuition behind Black-Litterman model portfolios". In: *Available at SSRN 334304*.

Meucci, Attilio (2010). *The Black-Litterman Approach: Original Model and Extensions. Shorter version in, The Encyclopedia of Quantitative Finance*.

Nguyen, Nguyet (2018). "Hidden Markov Model for Stock Trading". In: *International Journal of Financial Studies* 6.2, p. 36.

Schwarz, Gideon et al. (1978). "Estimating the dimension of a model". In: *The annals of statistics* 6.2, pp. 461–464.

Spreij, Peter (2001). "On the Markov property of a finite hidden Markov chain". In: *Statistics & probability letters* 52.3, pp. 279–288.

Tenyakov, Anton (2014). "Estimation of Hidden Markov Models and Their Applications in Finance". PhD thesis. The University of Western Ontario.

Zhang, Yingjian (2004). "Prediction of financial time series with Hidden Markov Models". PhD thesis. Applied Sciences: School of Computing Science.