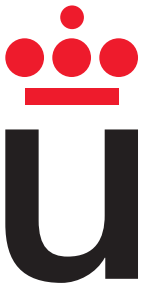


Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2022-2023

Trabajo Fin de Grado

DESARROLLO DE UNA BIBLIOTECA EN PYTHON PARA EL AJUSTE DE MODELOS OCULTOS DE MARKOV.

Aplicación al análisis de recesiones económicas.

Autor: Álvaro Fernández Moreno

Tutores: Alfredo Cuesta Infante
María Eugenia Castellanos Nueda

Agradecimientos

Me gustaría agradecer este trabajo a mi familia, que me han apoyado durante mi educación, mis amigos, especialmente los que he conocido durante la etapa universitaria y, finalmente a los profesores que me han acompañado durante mi aprendizaje. En particular, quiero dar las gracias a María Eugenia y Alfredo, por tratarme con especial atención y ayudarme a investigar algunos de los conceptos más complicados del trabajo, o incluso, yendo más allá.

Resumen

El desarrollo de modelos que replican situaciones del mundo real es fundamental para comprenderlo. Gracias al avance tecnológico, cada vez se crean modelos más sofisticados que permiten realizar razonamientos más precisos. En este trabajo, se aborda la teoría necesaria para la creación de modelos ocultos de Markov y se presenta una aplicación práctica de estos, en la que se tratan de identificar los estados de recesión de la economía estadounidense.

En el final del trabajo se ofrece una comparativa entre la biblioteca implementada y la biblioteca `TensorFlowProbability`, demostrando la versatilidad y exhaustividad del desarrollo.

Palabras Clave:

1. Modelo oculto de Markov.
2. Proceso estocástico.
3. Series Temporales.
4. Cadena de Markov.
5. Modelo mixtura.
6. *Expectation-Maximization*.
7. Viterbi.

Abstract

The development of models that replicate complex real-world situations is crucial for understanding them. Thanks to technological advancements, increasingly sophisticated models are being created, enabling more precise reasoning. This work addresses the necessary theory for creating hidden Markov models and delves into a practical application of these models in identifying recession states in the US economy.

At the end of the project, a comparison is provided between the implemented library and the `TensorFlowProbability` library, demonstrating the versatility and comprehensiveness of the development.

Keywords:

1. Hidden Markov Model.
2. Stochastic Process.
3. Time Series.
4. Markov Chain.
5. Mixture Model.
6. Expectation-Maximization.
7. Viterbi.

Índice general

Índice de figuras

Notación y Abreviaturas

La notación se irá introduciendo a lo largo del trabajo sin embargo podemos encontrar en la siguiente tabla los significados de cada símbolo y la página en la que se introducen:

Notación:

Símbolo	Significado	Página
$X^{(t)}$	Variable aleatoria indexada por un índice t	9
S	Conjunto de estados de la cadena de Markov (s_i será un estado particular)	9
A	Matriz de transición de estados (a_ij representa el valor particular de transitar de s_i a s_j)	12
a_{ij}	Probabilidad de transitar de s_i a s_j, valor de la matriz A	29
B	Matriz de emisión	16
$b_j(O_t)$	Probabilidad de observar O_t en el estado s_j, valor de la matriz **B**	29
$\boldsymbol{O}$	Vector de observaciones	13
π	Vector de probabilidades iniciales	29
λ	Parámetros de un modelo oculto de Markov ($\lambda = \{A, B, \pi\}$)	29
$\mathbf{s} = \{s_1, s_2, ..., s_T\}$	Vector de estados ocultos y desconocidos de la cadena oculta de Markov	29
α_t	Vector de probabilidad *forward* en el instante t	30
β_t	Vector de probabilidad *backward* en el instante t	31
γ_t	Probabilidad de estar en el estado s_i en un tiempo t	33
$\nu_t(j)$	Probabilidad de haber tomado el camino más probable hasta el estado s_j en el instante t	34
$\xi_t(i, j)$	Probabilidad de estar en el estado s_i en el instante t y en el instante $t+1$ estar en el estado s_j	37
$\hat{\alpha}_t$	Vector de probabilidad *forward* escalado	40
$\hat{\beta}_t$	Vector de probabilidad *backward* escalado	41
C_t	Constante de escalado para las probabilidades *forward*	40
D_t	Constante de escalado para las probabilidades *backward*	41
$L(\lambda \mid \mathbf{O})$	Función de verosimilitud de los datos dados los parámetros del modelo	29
$P(\mathbf{O} \mid \lambda)$	Función de verosimilitud sobre un λ fijo	30

Abreviaturas:

Siglas	Significado	Página
HMM	modelo oculto de Markov	3
i.i.d.	independiente e idénticamente distribuido	4
PIB o GDP	producto interior bruto	5
PNB	producto nacional bruto	5
GAN	red generativa adversaria	6
ETL	extracción, transformación y carga	20
AIC	*Akaike information criterion*	22
BIC	*Bayesian information criterion*	22
EM	*Expectation Maximization*	36
CDLL	log-verosimilitud de los datos completos	36
BCE	*Binary Cross Entropy*	49
TFP	*TensorFlow Probability*	53

Capítulo 1

Introducción

A lo largo del tiempo hemos intentado usar las matemáticas para modelar el mundo que nos rodea. Gracias a la revolución tecnológica que hemos vivido en el último siglo, hemos podido escalar los modelos matemáticos a niveles inimaginables. Hoy en día existen infinidad de casos en los que nos interesa utilizar un modelo que represente lo mejor posible los datos para poder tomar las decisiones óptimas. Y es en estas palabras "tomar decisiones" en las que nos vamos a centrar a lo largo de este documento, ya que los modelos de *machine learning* nos ayudan a capturar características inherentes de los datos.

Este trabajo introduce varios tipos de modelos probabilísticos y generalmente dinámicos en el tiempo que en la práctica ayudan a entender eventos pasados y predecir futuros. Más concretamente nos centraremos en los modelos ocultos de Markov (HMM) y las bases teóricas que los sustentan. Finalmente construiremos un modelo desde un punto de vista lógico, es decir, presentando un razonamiento lógico ante cada problema que surge.

1.1. Contexto y alcance

Es difícil identificar el origen de la estadística y el análisis de datos. Si bien es cierto que previamente en el siglo XVII se usaban teorías básicas de probabilidad para ganar en juegos de azar, el análisis de datos no se comenzó a usar hasta un tiempo después. El término "estadística" se acuñó en el siglo XVIII por la recolección de datos que realizaban los estados para conocer las características de una población. Posteriormente, se aplicó la estadística en distintos campos como la psicología, la sociología y otras ramas[20]. Sin embargo, la gran revolución no llegó hasta mediados del siglo pasado y comienzos de este siglo. El avance de la computación ha permitido realizar análisis más complejos y distanciarse de las formas más analíticas tradicionales.

1.1.1. ¿Qué es una serie temporal?

George Udny Yule y Norbert Wiener fueron los pioneros a la hora de aplicar los modelos estadísticos para analizar series temporales[35]. Una serie temporal no es más que una secuencia de datos que están ordenados cronológicamente. Estos datos se registran en unos intervalos de tiempos normalmente regulares, es decir, se registran en diferentes puntos en el tiempo, ya sea diariamente, semanalmente o anualmente. Entendamos que

es algo natural recoger datos para analizar el mundo que nos rodea, de esta forma tenemos acceso a un historial que nos permite conocer la evolución del sistema que estemos analizando. Gracias al estudio de las series temporales podemos examinar patrones, tendencias o ciclos que existan en los datos a lo largo del tiempo. Este estudio es lo que se conoce como inferencia, gracias a una serie de muestras a lo largo del tiempo logramos sacar conclusiones sobre una población. Más específicamente, al hacer inferencia sobre series temporales trataremos de comprender el comportamiento y predecir el futuro de los datos. A raíz de esto, podremos tomar decisiones informadas y adaptarnos a cualquier situación.

No obstante, a menudo encontramos dificultades a la hora de realizar inferencia, pues los datos pueden contener ruido, algunas variaciones puntuales o eventos atípicos. Además al tratarse de datos en el tiempo, puede existir una dependencia temporal con los valores pasados. De hecho, los primeros modelos que trataban de hacer inferencia en series temporales, alejados de la casuística del mundo real, trabajaban con la suposición de datos independientes e idénticamente distribuidos (*i.i.d.*). Con el tiempo, se han desarrollado técnicas más sofisticadas, como los modelos autorregresivos (ARIMA, GARCH, VAR), los modelos de descomposición de series temporales y los modelos de espacio estado, que permiten una mejor predicción y modelado de los datos temporales.

1.1.2. ¿Por qué elegimos HMM, qué nos ofrece esta técnica?

En la actualidad, los modelos existentes combinan varias técnicas para aunar y aumentar los beneficios que nos aporta cada una de ellas. Un caso particular de esto son, en efecto, los modelos ocultos de Markov pues mezclan diversos conceptos como las cadenas Markov, modelos de mixturas y técnicas de autorregresión. Estos modelos ofrecen un enfoque más complejo y sofisticado para la inferencia en series temporales, permitiendo capturar patrones ocultos y mejorar la capacidad de predicción en situaciones donde las observaciones son difíciles de modelar con enfoques más tradicionales.

1.1.3. Estado del arte

En los últimos años, con el auge del aprendizaje automático (*machine learning*), los modelos estadísticos para series temporales han evolucionado incorporando técnicas más avanzadas, como redes neuronales recurrentes, que han mejorado significativamente la precisión y la capacidad de predicción de estos modelos. Las redes neuronales recurrentes, junto con los modelos ocultos de Markov y los modelos autorregresivos mencionados anteriormente son los más utilizados para el análisis de series temporales. El paradigma actual ha demostrado que las redes neuronales o los modelos ocultos de Markov suelen ser una mejor opción a considerar para el análisis de series temporales[9].

Aunque las redes neuronales recurrentes permiten crear un modelo que ofrezca mayor adaptabilidad, que pueda inferir las características más abstractas, no siempre es la mejor solución. Los modelos ocultos de Markov son más eficientes y permiten ser entrenados con una cantidad pequeña de datos. También permiten una interpretación clara, ya que permiten observar la transición entre los estados a lo largo del tiempo y su probabilidad asociada.

1.2. Descripción del problema

Para poder construir modelos ocultos de Markov, hemos implementado una biblioteca en Python. La popularidad del ecosistema de Python en el análisis y procesamiento de datos ha ido en aumento en los últimos años, gracias a la simplicidad sintáctica del lenguaje y la facilidad de uso de las bibliotecas como numpy [7], scipy [31], matblotlib [10], JAX [4]. Además sobre estas bibliotecas se han creado otras más potentes que ofrecen una funcionalidad a un nivel de abstracción superior como pandas[28] para el análisis de datos, scikit-learn[19] para modelos de machine learning, Tensorflow[29] y Pytorch[18], estas últimas dos bibliotecas han sido desarrolladas por dos multinacionales tecnológicas como Google y Meta. A su vez, ambas han desarrollado sus correspondientes módulos de probabilidad, TensorflowProbability y Pyro[3].

Sin embargo, a pesar de que ambas bibliotecas contienen una implementación de los modelos ocultos de Markov, su uso puede estar limitado en términos de la comprensión detallada de los cálculos y el desarrollo existente detrás de estos modelos. Por ello, se han desarrollado otras bibliotecas alternativas como hmmlearn[8] y pomegranate[25].

En transcurso de este trabajo se ha preferido programar los modelos ocultos de Markov desde cero, centrándonos en la facilidad de uso y de comprensión. Nuevamente, este enfoque será gradual y se resolverán los problemas razonando de forma similar a la teoría. Para ello, se ha decidido implementar los algoritmos de forma constructiva en un notebook, de esta forma se pueden ejecutar las líneas de código a la vez que se lee el texto y se captan mejor los mensajes. Además, se proporcionará una clase en Python que recogerá toda la funcionalidad del notebook.

Posteriormente utilizaremos el código desarrollado para modelar un caso de uso particular en el que inferimos posibles estados de recesión en la economía. Asimismo, durante el desarrollo del caso de uso mostraremos cómo procesar los datos para hacer uso de la biblioteca desarrollada. Para resolver este problema particular es necesario incluir algo de contexto en cuanto al ámbito económico.

En primer lugar, la economía sigue un proceso cíclico, es decir, existe un periodo en el que la economía crece y otro periodo en el que se estanca o incluso decrece. Estos periodos se conocen como expansión y recesión respectivamente y se repiten de forma cíclica, aunque normalmente los periodos de expansión suelen durar más tiempo que las recesiones[11]. Existen múltiples causantes de una recesión como un cambio brusco en el mercado o en la cadena de producción, una deuda excesiva, especulación, incertidumbre provocada por la inflación o deflación, en general cambios que requieren de una reestructuración en el sistema económico.[1] Además, la economía, medida mediante el producto interior bruto, suele tener una tendencia alcista por diversos motivos como la política monetaria, el sistema capitalista en el que vivimos o los avances tecnológicos[34].

El producto interior bruto (PIB o del inglés *GDP*), mencionado anteriormente, es una medida del valor monetario de los bienes y servicios producidos en un país. Incluye diversos aspectos como el consumo, el gasto público, las inversiones empresariales y las exportaciones netas. Pero también hay muchas cosas que deja fuera, como el trabajo no remunerado, las ventas de bienes usados y, quizá lo más importante, el bienestar general. En general, los países con economías más fuertes y en crecimiento tienen niveles de vida más altos. Pero el PIB no es más que un indicador de cómo le van las cosas a la población[27]. Existen otros indicadores como el producto nacional bruto (PNB) que

incluye todos los productos o servicios generados fuera del territorio nacional o el índice de desarrollo humano (IDH) que mide una combinación entre la esperanza de vida, la educación y el producto interior bruto per cápita.

Aunque durante el siglo pasado surgieron multitud de teorías que trataban de modelar la economía, la realidad es que la complejidad del mundo en el que vivimos es tan elevada como para modelar y predecir con exactitud los eventos que vendrán en un futuro. En esta línea, en el trabajo se creará un modelo de la economía haciendo uso del PIB, y en concreto de la variabilidad del mismo.

1.3. Motivación Personal

La motivación principal del trabajo es estudiar un tipo de modelo de machine learning basado en un enfoque de estadística bayesiana, ya que la inferencia bayesiana ha sido el tema principal del trabajo expuesto en el TFG del grado en Matemáticas. A diferencia de la estadística clásica, el enfoque bayesiano es más natural pues considera unos conocimientos previos y a raíz de estos conocimientos, calcula unas creencias *a posteriori*. Además, el enfoque bayesiano suele representar cada parámetro del modelo como su propia distribución de probabilidad, lo que capta intrínsecamente los conocimientos sobre ese parámetro, mientras que los enfoques de máxima verosimilitud suelen representar cada parámetro del modelo como un valor.

También existe una motivación adicional en conocer un enfoque de aprendizaje no supervisado, pues todos los algoritmos aprendidos en la asignatura de *Inteligencia Artificial* han sido de entrenamiento supervisado. Realmente se dan multitudes de casos en los que no se disponen de datos de entrenamiento etiquetados y por lo tanto hay que cambiar el enfoque y trabajar únicamente con los datos disponibles. Los modelos ocultos de Markov, junto con otros modelos como los modelos de *clustering* o modelos generativos adversarios (GANs), usan este enfoque para analizar los datos y hacer inferencia a partir de estos.

Por último, existe una motivación de utilidad detrás de la predicción de una recesión. Existen numerosos beneficios detrás de conocer con exactitud el próximo periodo de recesión: el más evidente es la toma de decisiones, ya que puede ayudar a empresas, familias y gobiernos a controlar los riesgos existentes en sus propias economías y consecuentemente minimizar el impacto que esa recesión puede llegar a tener. Un ejemplo particular, podría ser la gran recesión de 2008, en la que la gente tenía unas carteras de activos altamente apalancadas[12], por ello si hubieran sabido que vendría una recesión podrían haberse anticipado y las consecuencias hubieran sido menores. En segundo lugar, existe un beneficio económico, ya que podremos predecir con facilidad los mercados financieros aprovechando las recesiones para invertir en activos a precios más bajos. Finalmente, podemos tener una planificación financiera, reduciendo gastos y ahorrando más dinero pera hacer frente a la recesión y un posible despido o disminución de ingresos.

1.4. Objetivos

Este trabajo tiene como objetivo principal diseñar e implementar una biblioteca que permita crear modelos ocultos de Markov. Además, tiene como segundo objetivo principal, resolver un problema específico de inferencia utilizando la biblioteca desarrollada

en Python. El problema particular a resolver se centra en el análisis de series temporales. Por lo tanto, para llevar a cabo estos objetivo, se han establecido unos sub-objetivos específicos:

1. Estudiar el estado del arte del análisis de series temporales.
2. Comprender los fundamentos teóricos de los modelos que se van a implementar.
3. Entender el uso de la programación dinámica para el cálculo de las probabilidades $forward$ y *backward*, o algoritmos como Viterbi.
4. Especificar los requisitos necesarios para construir un modelo oculto de Markov que pueda ser entrenado y haga predicciones.
5. Aprender todas las técnicas para estimar las funciones de interés de un modelo oculto de Markov (Algoritmos de aprendizaje no supervisado, Baum-Welch).
6. Estudiar la complejidad de los algoritmos.
7. Establecer posibles casos de uso en los que se pueda emplear la biblioteca para modelar un problema.
8. Proporcionar conclusiones y recomendaciones basadas en los resultados obtenidos, destacando las ventajas y limitaciones de los modelos ocultos de Markov en el contexto del análisis de series temporales, y sugiriendo posibles áreas de mejora o extensiones futuras del trabajo.

1.5. Estructura del documento

El documento constará de tres secciones. En la primera parte, *Fundamentos Teóricos*, veremos los conceptos teóricos más relevantes en relación a los modelos ocultos de Markov, viendo algunos ejemplos de algoritmos de entrenamiento no supervisado que usaremos en el estudio.

Después, una segunda parte en la que se diseñará una solución informática que permite construir y diseñar modelos ocultos de Markov, aplicando distintos algoritmos de aprendizaje. Y una tercera parte en la cual se aplica la solución creada para modelar la economía estadounidense y prever posibles recesiones. Esta sección sirve como una guía para el procesamiento de datos necesario antes de utilizar la biblioteca, así como para su posterior uso. Finalmente, expondremos los resultados obtenidos, sacaremos conclusiones de los mismos y propondremos trabajos futuros.

Capítulo 2

Fundamentos Teóricos

Para poder proponer y ajustar un modelo oculto de Markov que consiga identificar correctamente los estados de recesión, es necesario introducir unos conocimientos teóricos previos.

2.1. Proceso estocástico

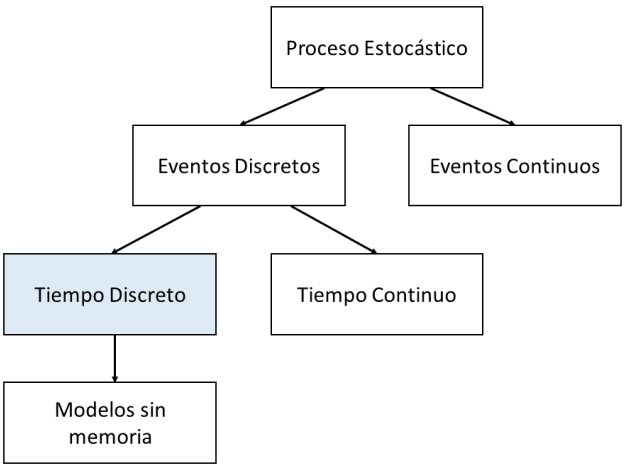


Figura 1: Esquema de los posibles procesos estocásticos.

Definición 1. *Un **proceso estocástico** es una colección de variables aleatorias* $\{X^{(t)}\}$*, indexadas por un índice* t*, con* $t \in T \subseteq \mathbb{R}$*. Donde* T *puede ser tanto un conjunto continuo como discreto.*

En otras palabras, un proceso estocástico es un modelo matemático que se utiliza para representar la evolución de una variable aleatoria en función de otra variable, generalmente el tiempo. En un proceso estocástico, cada variable aleatoria $X^{(t)}$ está asociada a un único elemento t del **conjunto de índices** T. El conjunto de variables aleatorias $\mathbf{X}$ se denomina **espacio de estados**, y están definidas en un espacio probabilístico (Ω, F, P), donde Ω es el conjunto de todos los posibles resultados de un experimento aleatorio.

Generalmente, llamaremos modelo estocástico a una descripción matemática de un sistema complejo "aleatorio" en la que se intenta representar la incertidumbre en las observaciones. Por ejemplo, consideremos un proceso discreto que varía con el tiempo: supongamos que tenemos dos estados $(S) = \{\text{frío}, \text{calor}\}$, entonces podríamos modelar la temperatura medida a lo largo del tiempo, en el tiempo inicial t_0, tenemos un estado $s_0 =$ frío, y en los siguientes días, $s_1 =$ frío, $s_2 =$ calor, $s_3 =$ frío, etc.

El primer matemático en presentar una definición precisa de los procesos estocásticos fue Khinchin[30]. Aunque la exploración de este tema se ha llevado a cabo durante siglos, el **proceso de Bernoulli** es posiblemente el ejemplo más sencillo de un proceso estocástico, tal como se mencionó anteriormente. Este proceso se caracteriza por tener temperaturas independientes entre estados. En un proceso de Bernoulli, observamos una secuencia de eventos binarios (éxito o fracaso) que son independientes en el tiempo. La probabilidad de éxito se mantiene constante a lo largo del tiempo. Un ejemplo común es lanzar una moneda equilibrada, en el cual existen dos posibles resultados: cara o cruz. La probabilidad de obtener cara en cada lanzamiento es de 0,5.

El primero en dar una definición matemática de los procesos estocásticos fue Khinchin[30]. Aunque se ha indagado mucho acerca de este tema, el **proceso de Bernoulli**, expuesto en el caso anterior, es posiblemente el ejemplo más sencillo de un proceso estocástico. En un proceso de Bernoulli tenemos una secuencia de eventos binarios (éxito o fracaso) independientes en el tiempo, en el que la probabilidad de éxito es constante a lo largo del tiempo. Otro ejemplo sería lanzar una moneda equilibrada, aquí tenemos dos resultados posibles, cara o cruz, y la probabilidad de obtener una cara es 0,5 en cada lanzamiento.

Por otro lado, tenemos procesos continuos como los *procesos de Poisson*. En este proceso estocástico se intenta modelar el número de eventos que ocurren al azar en el tiempo o espacio, teniendo una tasa promedio de ocurrencia . Por lo tanto, en cada instante de tiempo t_i, la variable aleatoria $X^{(t_i)}$ sigue una distribución Poisson(λ_i).

A modo de ejemplo, consideremos el número de llamadas telefónicas que recibe un centro de atención al cliente en un día determinado. Supongamos que el número promedio de llamadas que se reciben en un día es 50. Podemos modelar este proceso como un proceso de Poisson, donde tenemos una variable aleatoria que es el número de llamadas que reciben por día.

El proceso de Poisson se caracteriza por la propiedad de que los eventos ocurren de forma independiente y con una tasa promedio constante. En el ejemplo de las llamadas telefónicas, podemos suponer que cada llamada es un evento independiente y que la tasa promedio de llegada de llamadas es de 50 llamadas por día.

Entonces, podemos modelar el número de llamadas que se reciben en un día como una variable aleatoria con una distribución de $Poisson(\lambda = 50)$. La probabilidad de recibir k llamadas en un día determinado se puede calcular mediante:

$$P(X = k) = \frac{e^{-\lambda} * \lambda^k}{k!}$$

2.2. Modelos de mixtura

En la práctica, el proceso de Poisson que hemos descrito puede capturar la realidad de forma incorrecta. En muchas ocasiones, nos encontraremos que el número de llamadas no es constante a lo largo del día, es decir, existen unos intervalos de tiempo en los que la tasa de llamadas recibidas es más elevada y en otros momentos es muy escasa, como durante la noche. En este caso, podríamos asumir que el proceso de Poisson subyacente es una mixtura de varios procesos de Poisson, cada uno con una tasa de llamadas diferente.

Esto quiere decir que asumiríamos que el proceso de Poisson subyacente es una combinación de varios procesos de Poisson, cada uno con su tasa de llamadas concreta. En nuestro ejemplo, podríamos decir que el proceso de Poisson está compuesto por un proceso de Poisson con un valor de λ alto durante las horas pico y otro proceso de Poisson con un valor bajo durante el resto del día. Luego, para modelar el número de llamadas que se reciben en un día determinado, usaríamos un modelo de mixtura para combinar estos dos procesos de Poisson con diferentes pesos. Gracias a estos modelos podemos expresar la heterogeneidad de los datos y modelar la distribución de eventos que pueden ocurrir a diferentes tasas en diferentes momentos del día.

Los modelos de mixtura son una forma de trabajar con datos dispersos que generalmente tienen varias modas. Como hemos dicho, gracias a los modelos de mixturas podemos tener en cuenta la heterogeneidad de la población, sobre todo cuando una población está compuesta por varios grupos y cada grupo tiene una distribución distinta para la variable aleatoria que tratamos de estudiar.

En el libro *Hidden Markov Models for Time Series* [36] se incluye un ejemplo en el que dividen los clientes que compran cigarrillos en un estanco en tres grupos: fumadores, fumadores ocasionales y no fumadores. Argumenta que el número de paquetes de cigarrillos que compra cada grupo se distribuye Poisson, sin embargo la distribución completa será una distribución más dispersa que una Poisson o incluso multimodal ya que reflejará los distintos grupos de consumidores.

En la práctica, los modelos de mixturas son útiles para poder identificar los subgrupos existentes dentro de una población. Es decir, en lugar de conocer que existen los 3 grupos de fumadores, el modelo no "conoce" nada acerca de los clientes del estanco y gracias a un análisis a posteriori de los datos podremos identificar estos clientes. En nuestro problema tendremos una serie de variables latentes z que desconoceremos, estas variables no se observan directamente a diferencia de las variables observables. Volviendo al ejemplo de los clientes del estanco, las variables latentes será la categoría a la que pertenecen y variables observables podrían ser los hábitos de consumo.

2.3. Cadenas de Markov

Durante el proyecto usaremos las cadenas de Markov para modelar la secuencia de eventos del proceso estocástico y la probabilidad de que ocurra un evento dado.

Definición 2. *Una **cadena de Markov**, $\{X^{(t)}\}$ es una secuencia de variables aleatorias dependientes*

$$X^{(0)}, X^{(1)}, X^{(3)}, ..., X^{(t)},$$

tal que para cualquier tiempo, t, la distribución de probabilidad de $X^{(t)}$ depende únicamente de su último valor más reciente $X^{(t-1)}$.

Este tipo de modelos son llamados modelos sin memoria, ya que los resultados futuros únicamente dependen del estado actual. También podremos definir la probabilidad de pasar de un estado $X^{(t-1)}$ a otro $X^{(t)}$ en el siguiente instante, conocido como kernel de transición:

Definición 3. *También se define el **kernel de transición** o **kernel Markoviano**, K, como la distribución de probabilidad condicionada:*

$$X^{(t)}|X^{(0)}, X^{(1)}, ..., X^{(t-1)} = X^{(t)}|X^{(t-1)} \sim K(X^{(t)}, X^{(t-1)})$$

En muchas ocasiones, las cadenas de Markov tenderán a una distribución límite conocida como **distribución estacionaria**, f. Es decir, a partir de un tiempo t, las siguientes variables aleatorias $X^{(t+1)}, X^{(t+2)}, ...$, tienen la misma distribución

En los casos de tiempo discretos, las cadenas de Markov se suelen representar mediante grafos. Por ejemplo, veamos el siguiente ejemplo de una cadena de Markov con tres estados:

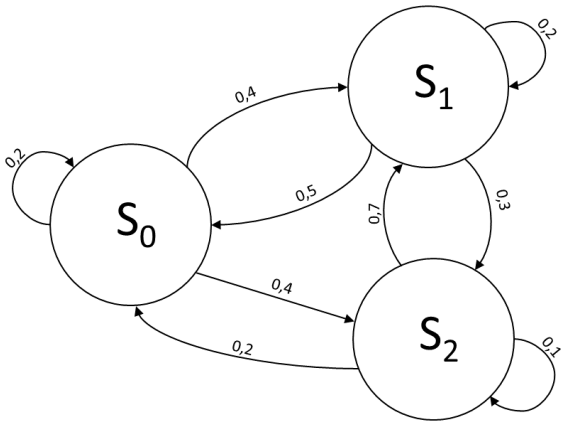


Figura 2: Grafo con las probabilidades de transición entre estados de una cadena de Markov con tres estados.

Estas transiciones también se pueden escribir de forma matricial, recogiendo en cada fila el Kernel de transición de la variable $X^{(t)}$. Como esta cadena de Markov es discreta, la función de distribución será categórica.

$$A = \begin{pmatrix} 0{,}2 & 0{,}4 & 0{,}4 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}7 & 0{,}1 \end{pmatrix}$$

A raíz de esto, podemos preguntarnos, ¿cuál será la probabilidad de pasar por una secuencia de estados?, o ¿cuál es la probabilidad de que en un instante t_j estemos en el estado s_j sabiendo que en el instante t_i estábamos en el estado s_i?

En respuesta a la primera pregunta, gracias a la propiedad de Markov que nos indica que el estado actual únicamente depende del estado anterior, podemos observar lo siguiente:

$$\begin{aligned} P(X^{(n)}, X^{(n-1)}, ..., X^{(0)}) &= P(X^{(n)}|X^{(n-1)}, ..., X^{(0)}) \\ &\quad \cdot P(X^{(n-1)}|X^{(n-2)}, ..., X^{(0)}) \cdot \ \ldots \ \cdot P(X^{(1)}|X^{(0)}) \\ &= P(X^{(n)}|X^{(n-1)}) \cdot P(X^{(n-1)}|X^{(n-2)}) \cdot \ \ldots \ \cdot P(X^{(1)}|X^{(0)}) \end{aligned}$$

Para calcular la probabilidad de ir del estado s_i en el instante t_i a otro estado s_j en una cantidad finita de instantes n, podemos hacer uso de las ecuaciones de Chapman-Kolmogorov:

$$\begin{aligned} P_{s_i,s_j}(t_i, t_i+n) &= P(X^{(t_{i+n})} = s_j | X^{(t_i)} = s_i) \\ &= \sum_{z \in S} P(X^{(t_{i+n})} = s_j | X^{(t_{i+(n-1)})} = z, X^{(t_i)} = s_i) \cdot P(X^{(t_{i+(n-1)})} = z | X^{(t_i)} = s_i) \\ &= \sum_{z \in S} P(X^{(t_{i+n})} = s_j | X^{(t_{i+(n-1)})} = z) \cdot P(X^{(t_{i+(n-1)})} = z | X^{(t_i)} = s_i), \end{aligned}$$

donde S es el conjunto de estados. La idea es ir partiendo las transiciones en transiciones de un único paso y así conseguimos reducir la complejidad del cálculo a la probabilidad de ir del estado s_i a un estado cualquiera s_k en $n-1$ tiempos ($P(X^{(t_{i+(n-1)})} = s_k | X^{(t_i)} = s_i)$). No obstante, estos cálculos recursivos son fáciles de hacer ya que son potencias de la matriz de transición A. Por lo que si queremos calcular $P_{s_i,s_j}(t_i, t_i+n)$ basta con considerar la n-ésima potencia en caso de que la cadena sea homogénea[1]:

$$P_{s_i,s_j}(t_i, t_i+n) = (A^n)_{s_i,s_j} \text{ (cadena homogénea)}$$
$$P_{s_i,s_j}(t_i, t_i+n) = (A^{t_i+n})_{s_i,s_j} \text{ (cadena no homogénea)}$$

Gracias a estas ecuaciones también podríamos calcular la probabilidad que existe de quedarnos en un mismo estado durante exactamente d instantes. Introduciendo nueva notación, tendremos una secuencia de observaciones:

$$\boldsymbol{O} = \{X^{(t)} = s_i, X^{(t+1)} = s_i, ..., X^{(t+d)} = s_i, X^{(t+d+1)} = s_j\}$$

Entonces, suponiendo que la cadena es homogénea:

$$P(\mathbf{O} \mid X^{(t)} = s_i) = P(X^{(t+1)} = s_i | X^{(t)} = s_i)^{(d-1)} \cdot (1 - P(X^{(t+1)} = s_i | X^{(t)} = s_i))$$

[1]En el cálculo de esta probabilidad, hemos supuesto que la cadena es **homogénea**, es decir que la probabilidad de cambiar del estado s_i al estado s_j en un instante, no depende del tiempo en el que se encuentra la cadena:

$$P(X^{(t_i+1)} = s_j | X^{(t_i)} = s_i) = P(X^{(t_j+1)} = s_j | X^{(t_j)} = s_i)$$

para cuales quiera $t_i, t_j > 0$. En caso contrario se dice que la cadena es **no homogénea**:

Hay que mencionar que también existen cadenas de Markov de tiempo continuo (CTMC), en estos casos la probabilidad de transitar a otro estado depende del estado actual y del tiempo trascurrido en este estado (conocido como tiempo de espera, *holding time*).

2.3.1. Propiedades y tipos de cadenas de Markov

Existen una serie de propiedades y atributos que caracterizan a las cadenas de Markov:

Definición 4. *Se define el* ***periodo*** *de un estado $s_i \in \{X^{(t)}\}$ como:*

$$d(s_i) = mcd(n \geq 1 \mid P_{s_i,s_i}(t, t+n) > 0)$$

por lo que si $d(x) = 1$ entonces decimos que x es un estado ***aperiódico****, es decir que no se va a volver a reproducir. Sin embargo si $d(s_i) = k \geq 2$ decimos que s_i tiene un periodo k. También decimos que una cadena de Markov es aperiódica si todos sus estados son aperiódicos.*

Definición 5. *Una cadena de Markov $\{X^{(t)}\}$ se dice que es* ***irreducible*** *si se cumple que:*

1. *Desde cualquier estado $x \in \{X^{(t)}\}$ se puede acceder a cualquier otro estado.*
2. *Todos los estados se comunican entre sí.*
3. *Todos los estados pertenecen a una clase de comunicación cerrada.* [2]

por lo que en resumen, una cadena de Markov será irreducible si el kernel K nos permite movernos libremente entre todos los estados.

Antes hemos definido de palabra el concepto de distribución estacionaria, para profundizar en este concepto es necesario definir las probabilidades estacionarias δ. Este vector de probabilidades de tamaño $N = cardinal(S)$ nos indica las probabilidades de estar en cada estado de la cadena de Markov cuando se alcanza el equilibrio. Estas probabilidades satisfacen la siguiente ecuación:

$$\delta = \delta A$$

donde A es la matriz de transición de la cadena de Markov. Como ya hemos mencionado, sólo las cadenas de Markov que son irreducibles y aperiódicas tienen una distribución estacionaria y, en caso de que la distribución estacionaria exista, no tiene por qué ser única.

El hecho de que una cadena de Markov sea estacionaria tiene distintas implicaciones, una de las más importantes es la **recurrencia**. Una cadena es recurrente si puede volver a visitar un estado un número infinito de veces, en cambio decimos que un estado es **transitorio** en caso contrario.

Adicionalmente, existe un concepto fundamental en la aplicación de las cadenas de Markov llamado **ergodicidad**.

[2] Se dice que dos estados i, j están comunicados si existe un camino en el grafo que una i con j, de esta forma podemos establecer una relación de equivalencia si un estado i está comunicado con un estado j y viceversa. Por lo tanto, podemos hablar de la clase de equivalencia de i y recibe el nombre de clase de comunicación.

Definición 6. *Se dice que una cadena de Markov es ergódica si es posible saltar de cada estado a otro estado cualquiera (no necesariamente en un paso). Es decir, una cadena de Markov es ergódica si y sólo si,*

$$P_{s_i,s_j}(t, t+n) > 0$$

para cuales quiera $s_i, s_j \in \{X^{(t)}\}$, $t, n > 0$.

Esto implica que la distribución límite de la cadena de Markov $\{X^{(t)}\}$ no depende del estado inicial $X^{(0)}$ del que partamos.

2.4. Modelos Ocultos de Markov

Aunque la utilización de cadenas de Markov para modelar situaciones del mundo real es una idea prometedora, en la práctica nos encontramos con situaciones en las que existen estados Y que observamos pero que están influenciados por otros estados latentes X. Es en este contexto es donde surgen los modelos ocultos de Markov (HMM), los cuales se enfocan en aprender o predecir los estados X a partir de las observaciones Y.

Consideremos el siguiente ejemplo, en el que queremos predecir el estado de humor de nuestro amigo en función de la camiseta que lleva un día particular. Por lo que tendremos:

- Conjunto de datos o estados observables (Y) = {Amarilla (A), Roja (R)}
- Conjunto de estados ocultos (X) = {Feliz (F), Triste (T)}

Los modelos ocultos de Markov, cumplen con la propiedad de Markov, es decir que el valor del estado actual depende únicamente del estado anterior. Además, existen unas probabilidades de transitar de un estado a otro en un instante de tiempo. Para el ejemplo anterior podemos ilustrarlo con el siguiente diagrama:

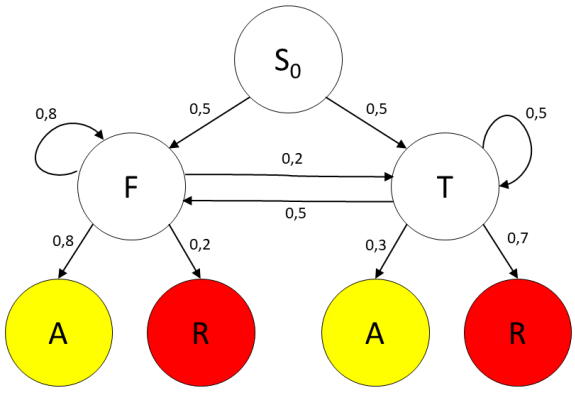


Figura 3: Grafo con las probabilidades de transición entre estados. Existe un estado inicial S_0 que nos establece las probabilidades de que nuestro amigo esté feliz o triste, el primer día que le conocimos (cuando no tenemos información *a priori*). Los estados F y T, son los que no podemos observar directamente, y los estados observables A y R indican el color de la camiseta que vemos.

Este grafo se puede expresar mediante dos matrices de probabilidades, la primera contendrá las **probabilidades de emisión** que se refiere a la probabilidad de que un estado oculto X emita una observación observable Y. Es decir, dada la ocurrencia de un estado oculto en un momento dado, ¿cuál es la probabilidad de observar una salida particular en ese instante?

$$\text{Matriz de emisión: } B = \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}3 & 0{,}7 \end{pmatrix}$$

Por otro lado, la probabilidad de transición es la probabilidad de pasar de un estado oculto a otro estado en el siguiente momento de tiempo.

$$\text{Matriz de transición: } A = \begin{pmatrix} 0{,}8 & 0{,}2 \\ 0{,}5 & 0{,}5 \end{pmatrix}$$

Por lo tanto, poder predecir si tu amigo está triste o feliz en base al color de la camiseta parece cuanto menos difícil ya que depende del estado en el que nos encontremos, y aunque seamos capaces de conocer el estado actual del día de hoy, difícilmente podremos averiguar el estado en el que se encontrará mañana, ya que existe una posibilidad de que transitemos al estado opuesto o que observemos un resultado menos favorable para el estado en el que estamos.

A modo de adelanto, el objetivo final de cualquier modelo oculto de Markov, es conocer la secuencia de estados ocultos que dan lugar a los estados que observamos en la vida real. Esto es, por ejemplo, si nuestro vector de observaciones es $\boldsymbol{O} = \{O_1 = R, O_2 = R, O_3 = A\}$, nos gustaría saber qué secuencia de estados ocultos son los más probables. Equivalentemente, buscamos el vector de estados ocultos $\{m_1, m_2, m_3\}$ que maximicen la siguiente probabilidad:

$$P(X^{(1)} = m_1, X^{(2)} = m_2, X^{(3)} = m_3; O_1 = R, O_2 = R, O_3 = A,)$$

Esta probabilidad se puede calcular por fuerza bruta pero su complejidad computacional es alta ($\mathrm{O}(N^T)$), o con ayuda de algoritmos de programación dinámica como el algoritmo de Vitrebi. Este algoritmo lo veremos en un futuro junto con el desarrollo lógico que hay que seguir para modelar un problema del mundo real mediante un modelo oculto de Markov.

2.5. Programación Dinámica

Este paradigma de programación consiste en dividir un problema en subproblemas con el objetivo de reducir el tiempo de ejecución de un algoritmo. En esencia, la programación dinámica consiste en descomponer un problema en subproblemas más pequeños, utilizando la solución de un subproblema para resolver el siguiente.

Una técnica empleada en programación dinámica consiste en el uso de **subestructuras óptimas**, esto es, que la solución óptima al problema principal se puede construir mediante soluciones óptimas de subproblemas más pequeños. En otras palabras, si se tienen subproblemas que comparten una solución óptima, entonces se dice que el problema tiene una subestructura óptima. Por otro lado, si la solución a los subproblemas (o al

menos parte de ella) se puede usar varias veces en un algoritmo, entonces se dice que el problema original tiene **subproblemas superpuestos**. Esto da lugar a una técnica usada en programación dinámica y conocida como **memoización**. Consiste en evitar la resolución repetida de subproblemas superpuestos, guardando solución de un subproblema ya calculado, y en el momento que nos encontremos con el mismo subproblema, simplemente recuperamos la solución de la memoria.

Existen dos enfoques para resolver problemas dentro de la programación dinámica: el enfoque **descendente** (*top-down*) y el enfoque **ascendente** (*bottom-up*). Ambos enfoques utilizan la idea de dividir un problema en subproblemas más pequeños y resolverlos de manera incremental, pero difieren en la forma en que se resuelven los subproblemas y se combinan las soluciones.

El enfoque descendente hace uso de la memoización. Comienza formulando una visión general del problema y dividiendo el problema original en subproblemas, y estos se resuelven guardando las soluciones por si fueran necesarias nuevamente. Luego, se combinan las soluciones de los subproblemas para obtener la solución del problema original. La recursión es un ejemplo de enfoque descendente, pues llamamos a la función hasta llegar a un caso base, además gracias a la memoización podemos incluir los resultados en las llamadas recursivas para evitar repetir cálculos innecesarios. Recibe el nombre de descendente porque empezamos con una imagen general y a raíz de esto vamos segmentando el problema en subproblemas.

Por otro lado, el enfoque ascendente trata de resolver todos los subproblemas posibles de antemano y después se usan para construir la solución de un problema mayor. Esta solución suele ser más eficiente en memoria, sin embargo tiene mayor dificultad a la hora de diseñar el algoritmo.

Después veremos cómo se hace uso de la programación dinámica para mejorar la eficiencia de los algoritmos. En concreto, veremos pseudocódigos (Algoritmo 1, Algoritmo 2, Algoritmo 3) que ilustran a la perfección las técnicas mencionadas anteriormente.

Capítulo 3

Diseño de la Biblioteca Software

El diseño de una solución es un procedimiento necesario para crear un sistema que cumpla con los requisitos y expectativas del usuario. En este caso realizaremos una biblioteca que permita modelar situaciones del mundo real con modelos ocultos de Markov.

En este capítulo, se describirá el diseño de la solución propuesta, incluyendo la arquitectura general, los requisitos necesarios para poder realizar inferencia a partir los datos y estimar los estados ocultos. Además, se proporcionarán diagramas de clases para ilustrar la estructura de la biblioteca y se discutirán los posibles límites y ampliaciones de la solución. En definitiva, este capítulo es una guía completa para entender cómo se ha diseñado la solución y cómo se pueden utilizar las diferentes funcionalidades de la biblioteca.

3.1. Generalidades de la biblioteca.

La solución propuesta es una biblioteca que permita realizar inferencia utilizando modelos ocultos de Markov. En particular, la biblioteca debe proporcionar un conjunto de herramientas y funcionalidades que permitan al usuario manipular los HMM de forma eficiente.

Dado que el objetivo principal de la solución es realizar inferencia a partir de datos, es necesario que el usuario final tenga conocimientos mínimos. En concreto, se requiere que el usuario procese y limpie los datos de forma adecuada para que la biblioteca trabaje con ellos y produzca buenos resultados. Además, el usuario necesitará un conocimiento mínimo acerca de los modelos de mixturas, los modelos ocultos de Markov descritos en el capítulo anterior 2 y los algoritmos que comentaremos durante la construcción de la biblioteca (sección 3.4).

Se ha decidido implementar la biblioteca en Python por varias razones. En primer lugar, Python es uno de los lenguajes más utilizados para el análisis de datos y *machine learning*. Una biblioteca de Python puede ser adoptada y utilizada por los usuarios fácilmente. Además, la biblioteca puede integrarse en módulos más extensos de *machine learning* o en *software* de terceros que requieran una herramienta para modelar un modelo de mixtura. Otro motivo es la flexibilidad, ya que la biblioteca puede ser modificada y adaptada a casos de uso más específicos.

En cuanto a la interfaz de usuario, se ha decidido no desarrollar una específicamente en este trabajo, ya que la biblioteca está diseñada para ser usada por terceros. No obstante,

se podría realizar una interfaz para facilitar el procesamiento de datos, en particular, la unión de varios conjuntos de datos y la visualización de los mismos. Sin embargo, esto queda fuera del ámbito de este trabajo.

Aunque no se incluye una interfaz de usuario, se ha creado un cuaderno de Jupyter que proporciona un ejemplo de la implementación de la biblioteca. Esta herramienta, puede ser útil para informáticos a la hora de visualizar los datos antes de utilizar la biblioteca y por lo tanto actuar como interfaz de usuario.

3.1.1. Tratamiento de datos.

Tal como hemos mencionado, es necesario que el usuario realice un tratamiento de los datos antes de ser usados por la biblioteca. Si se introducen valores incorrectos, como datos anómalos o nulos, la biblioteca no funcionará adecuadamente. Es por esto que es imprescindible realizar un análisis descriptivo de los datos antes de realizar cualquier tipo de inferencia. Este análisis implica explorar los datos en busca de elementos perdidos, anómalos, duplicados o con un formato distinto. Después de realizar este análisis tendremos que realizar una curación de los datos. En general el procedimiento de procesamiento de los datos para asegurar la calidad de los mismos antes de ser utilizados en el modelo será el siguiente:

ETL

El procedimiento de extracción, transformación y carga, o las siglas ETL del inglés *Extraction, Transform and Load*, consiste en extraer los datos necesarios de una o varias fuentes de datos. Normalmente estos están "sucios", es decir, tienen valores nulos, duplicados, en formatos inadecuados, sin sentido, *outliers*, etc. Consecuentemente tenemos que transformar los datos para trabajar adecuadamente con ellos en un formato que sea cómodo y sean fáciles de analizar. Este proceso de transformación también incluye el proceso de unificación de datos de distintas fuentes, ya que generalmente accederemos a distintos recursos para posteriormente trabajar con ellos conjuntamente. Finalmente este conjunto de datos es almacenado en una base de datos para que sean accesibles en un futuro.

En los últimos años se ha hecho tan popular este modelo que existen empresas dedicadas exclusivamente al procesamiento de datos. Aquellas empresas que manejan un gran volumen de información, suelen guardarla en un *data warehouse*. Un *data warehouse* es un sistema de almacenamiento de datos diseñado específicamente para facilitar la consulta y el análisis de grandes volúmenes de información. Al utilizar un *data warehouse*, los datos procesados durante el proceso ETL se organizan y estructuran de manera eficiente, lo que permite un acceso rápido y sencillo para su posterior uso. Este concepto llevado al extremo se conoce como *data lake*. A diferencia de un *data warehouse*, en un *data lake* no se almacena la información de forma estructurada, pues los datos ya se transformarán cuando sean útiles.

Aunque el proceso natural de trabajo sea el ETL, a veces resulta más cómodo invertir las etapas de transformación y carga dando una mayor flexibilidad en el procesamiento de datos ya que podemos modificarlos posteriormente en función de nuestros intereses. Sin embargo, una gran desventaja del sistema ELT (*Extraction, Load and Transform*) es la necesidad de un mayor espacio de almacenamiento, ya que los datos al almacenarse en

su formato original suelen contener información redundante para el análisis que se va a llevar a cabo.

Normalmente, los procesos de extracción y carga de la información son bastante directos. El proceso de extracción suele ser directo de una fuente de datos como una base de datos (relacional o no relacional), un archivo (CSV, Excel, etc.), un sistema ERP (Enterprise Resource Planning) o llamadas APIs a una fuente externa. También existen otros métodos más agresivos de extraer datos como *web-scrapping*. Sin embargo, el procedimiento más complejo es el de transformar la información en un conjunto de datos estructurado y con sentido. Podemos distinguir algunas de las tareas principales en el proceso de transformación de datos:

1. Limpieza de datos: se eliminan valores atípicos, duplicados, inconsistentes o nulos.
2. Adecuación de datos: dado que los datos provienen de múltiples fuentes con formatos diferentes, es necesario adaptarlos para mantener la coherencia y consistencia.
3. Etapa operacional: se aplican distintas operaciones y reglas necesarias para el análisis posterior. Este proceso puede incluir tareas de normalización de datos, derivaciones y cálculos.

3.2. Análisis de los requisitos

Para discutir los requisitos necesarios para construir la biblioteca vamos a distinguir varios casos de uso y caminos alternativos a la hora de usar la biblioteca. Una vez comprendamos cómo se va a usar la biblioteca, extraeremos de aquí los requisitos del software necesarios.

<table>
<tr><td colspan="2">Flujo de eventos</td></tr>
<tr><td colspan="2"><u>Camino básico</u> del caso de uso “Definir un modelo dados los parámetros y estimar los estados ocultos pasados y futuros.”</td></tr>
<tr><td colspan="2"><u>Descripción</u>:
El camino básico comienza cuando el usuario tiene unos datos procesados de unas observaciones que siguen una distribución de probabilidad definida.</td></tr>
<tr><td>ACTOR</td><td>SISTEMA</td></tr>
<tr><td>1. Inicializa la clase introduciendo las observaciones indexadas por una variable temporal, una matriz de probabilidades iniciales, una matriz de transiciones y una matriz de distribuciones con una entrada por cada estado latente.</td><td>2. Inicializa la clase HMM.</td></tr>
<tr><td>3. Calcula la probabilidad de observar el vector $\boldsymbol{O}$ dados los parámetros del modelo λ ($P(\boldsymbol{O} \mid \lambda)$).</td><td>4. Devuelve $P(\boldsymbol{O} \mid \lambda)$.</td></tr>
<tr><td>5. Entrega otro conjunto de datos estimar la secuencia de estados $\boldsymbol{s}$ más probable dadas unas observaciones $\boldsymbol{O}$ y los parámetros λ.</td><td>6. Devuelve una matriz con los estados más probables y la confianza de los resultados obtenidos.</td></tr>
<tr><td>7. Pide una predicción de los estados para los siguientes t instantes temporales.</td><td>8. Devuelve una matriz con los estados más probables y la confianza de los resultados obtenidos.</td></tr>
<tr><td colspan="2"><u>Caminos alternativos</u></td></tr>
<tr><td colspan="2">Evento 1. El actor puede no introducir la matriz de probabilidades iniciales, la matriz de estados y la matriz de distribuciones.</td></tr>
</table>

Tabla 1: Caso de uso para crear un modelo y predecir tanto estados presentes como futuro.

En primer lugar consideraremos el flujo de eventos del camino básico para crear un modelo y predecir tanto estados presentes como futuros. El actor será cualquier usuario que utilice la biblioteca. Hemos distinguido un caso alternativo en el primer evento, ya que el usuario puede no conocer el número de estados subyacentes en el modelo oculto de Markov. En ese caso, para continuar con el camino básico, el sistema tendrá que emplear algún criterio, AIC o BIC, para encontrar la cantidad de parámetros necesarios e inicializar la matriz de probabilidades iniciales, la matriz de estados y la matriz de distribuciones. No obstante, sí se requiere que el usuario especifique la distribución de las variables de emisión.

Flujo de eventos	
Camino básico del caso de uso "Entrenar un modelo".	
Descripción: El camino básico comienza una vez que el usuario ya ha creado una instancia de la clase HMM.	
ACTOR	SISTEMA
1. Introduce un conjunto de datos y pide que se entrene el modelo, ajustándolos a una parte de ese conjunto de datos.	2. Hace una división del conjunto de datos y entrena el modelo usando el algoritmo indicado por el usuario.

Tabla 2: Caso de uso para entrenar un modelo.

A raíz de esto podemos identificar tanto requisitos funcionales, que describen los servicios que debe proporcionar el software, como no funcionales, que describen la operabilidad de la solución.

Requisitos Funcionales.

- El sistema tiene que poder recibir datos procesados, indexados por una variable temporal.
- El sistema permite definir la matriz de probabilidades iniciales, la matriz de transición de estados y una matriz con las distribuciones de los estados latentes.
- El sistema permite trabajar con distribuciones de emisión continuas o discretas, pero únicamente distribuciones de transición discretas. Es decir, no se podrá definir un modelo oculto de Markov en el que los estados latentes sean continuos.
- El usuario podrá indicar el número de estados latentes que tendrá la cadena oculta de Markov. Esto se hará mediante la matriz de transición de estados, es decir, tomará el tamaño de la misma.
- El sistema podrá inicializar los valores de la matriz de probabilidades iniciales y matriz de transición de estados en caso de que el usuario no la introduzca. En este caso, se calculará el valor del AIC o BIC para determinar el número de estados e inicializará las matrices con un valor aleatorio.
- El sistema podrá calcular la verosimilitud del modelo y la log-verosimilitud.
- El sistema facilitará la modificación del modelo, es decir, el usuario podrá en cualquier momento cambiar los valores de la matriz de probabilidades iniciales, matriz de transición o matriz de distribuciones de emisión.
- El sistema tendrá la capacidad de hacer predicciones de un conjunto de datos y predicciones futuras, indicando la confianza de los resultados. El sistema devolverá un vector de tuplas del mismo tamaño que el recibido, donde se indiquen los estados más probables y los resultados obtenidos.

- El usuario podrá entrenar el modelo. Para ello, el usuario entregará el conjunto de datos sobre el que se quiere entrenar el modelo junto con otros parámetros adicionales. Entre los parámetros adicionales encontraremos:
  - El algoritmo empleado en el entrenamiento.
  - La proporción que se destinará del conjunto de datos para el entrenamiento, frente a valorar el ajuste del modelo de mixtura[1]. Por ejemplo, recibir un valor de 0,6, implicará que el 60 % de los datos se usarán para entrenar el modelo y el 40 % restante para la validación del ajuste.

Requisitos No Funcionales.

- El sistema será una biblioteca versátil y fácil de usar, que acepta datos en forma de lista o `numpy.array`.
- El sistema buscará la eficiencia computacional y permitirá trabajar con grandes volúmenes de datos.
- El sistema implementará tests que aseguren la robustez de la solución desarrollada.

3.3. Análisis del diseño

Dentro de la arquitectura de la solución, necesitaremos una interfaz que gestione todas las tareas que pide el usuario, como hemos señalado en los casos de uso (Tabla 1, Tabla 2) estas son inicializar el modelo, calcular distintas probabilidades, gestionar los parámetros, realizar predicciones y realizar un entrenamiento.

Además necesitaremos otra clase que permita diferenciar entre las distintas distribuciones de emisión, esto es necesario ya que el cálculo de las probabilidades *forward*, *backward* y el propio entrenamiento dependen directamente de la distribución de emisión. Por ello, será necesario que existan casos particulares que hereden de la clase principal. Más concretamente, se pueden ver en el siguiente diagrama de clases la estructura adoptada:

3.3.1. Diagramas de Clases

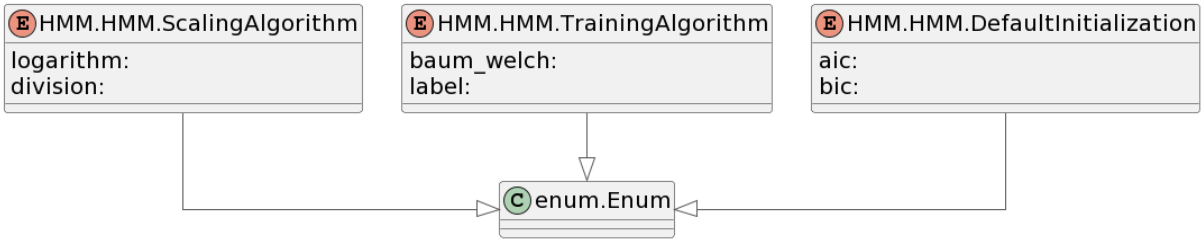


Figura 4: Diagrama de clases auxiliares para la construcción de un HMM.

[1]Nótese que el usuario podrá tener el conjunto de datos ya dividido. En ese caso, podrá validar el ajuste del entrenamiento indicando que se quiere entrenar con el 100 % de los datos y luego calcular la verosimilitud del conjunto de datos destinados al entrenamiento.

Figura 5: Diagrama de clases referente a la estructura de modelos ocultos de Markov.

A continuación, veremos cómo esta estructura cumple con los requisitos establecidos en la sección 3.2 y permite la realización de los casos de uso (Tabla 1 y Tabla 2).

Como podemos observar en la Figura 5, la clase HMM permite al usuario una abstracción a la hora de construir un modelo oculto de Markov. Como veremos en la siguiente sección 3.4, existen tres problemas principales que hay que resolver: calcular la verosimilitud de los datos, realizar predicciones de los estados, tanto instantes pasados como futuros, y finalmente entrenar el modelo. Como podemos ver estas tres tareas son abordadas por la clase principal `HMM`. Adicionalmente, se ha incluido la clase `CategoricalHMM` que, aunque no se vaya a usar en el resto del trabajo, permite ver las diferencias en la sobreescritura de los métodos abstractos entre las distintas clases que heredan de la clase padre.

En lo relativo al cumplimiento del primer requisito, la implementación realizada permite cambiar el vector de observaciones en varios instantes del flujo de eventos. Por ejemplo, al inicializar una instancia de la clase `HMM` podremos introducir un vector de observaciones, o al calcular la verosimilitud, o entrenar el modelo. Nótese el uso del tipo genérico `Any`, esto es porque las observaciones no son necesariamente un vector de dobles, en particular, en el caso discreto de una distribución categórica tendremos una lista de estados que podrán estar representados por un entero o una cadena de caracteres.

En segundo lugar, hay que comentar que existen tres formas de inicializar un objeto de la clase `HMM`:

1. Podremos inicializarlo usando los parámetros, λ, del modelo, esto es, la matriz de probabilidades iniciales, la matriz de transición y los parámetros de las distribuciones de emisión (en el caso de `NormalHMM` necesitaremos la media y la desviación estándar).

2. Podremos inicializarlo usando el número de estados que tendrá la cadena oculta de Markov y el vector de observaciones. En este caso, los parámetros se inicializarán de manera aleatoria, esto implica que el modelo no estará bien ajustado a los datos y por tanto, las predicciones serán imprecisas.

3. Finalmente, podremos inicializarlo usando únicamente un vector de observaciones, en este caso se buscará el modelo que mejor se ajuste a los datos. Esta evaluación se hará usando las métricas BIC y AIC.

En lo relativo al cálculo de la verosimilitud y de las probabilidades relacionadas con el entrenamiento, el usuario podrá especificar mediante el enumerado `ScalingAlgorithm` si prefiere los cálculos usando factores de escalado o un escalado logarítmico.

Respecto a las predicciones, hemos distinguido los siguientes casos:

- Se podrá realizar una decodificación global de la cadena de estados más probable dadas unas observaciones. Como hemos explicado en la teoría, este procedimiento se calcula usando el algoritmo de Viterbi. En la función `viterbi`, devolveremos la cadena más probable junto con su probabilidad. Adicionalmente, se añade un parámetro `state_probs` en la llamada a la función `viterbi` que indica si se quieren devolver también las probabilidades asociadas a cada estado. Recordemos que la probabilidad máxima de estar en el estado s_i en el tiempo t puede no coincidir con el resultado en el tiempo t de la cadena más probable.

- Se podrán obtener las distribuciones condicionales asociadas a cada instante t. Estas distribuciones permitirán realizar muestras en cada periodo temporal y también podremos comparar estas con el vector de observaciones dado. La distribución condicional es la siguiente:

$$Pr(X^{(t)} = x|\mathbf{O}_{(-t)}), \tag{3.3.1}$$

  donde el vector $\mathbf{O}_{(-t)}$, es el vector de observaciones en los instantes distintos de t:

$$\mathbf{O}_{(-t)} \equiv (O_1, ..., O_{t-1}, O_{t+1}, ..., O_T)$$

- En esta misma línea, el método `forecast_dist` nos devolverá una lista de distribuciones en h tiempos futuros. Estas distribuciones se conocen como distribuciones predictivas y vienen dadas por:

$$Pr(X^{(T+h)} = x|\mathbf{O}) \tag{3.3.2}$$

- Finalmente, podremos obtener predicciones de estados futuros gracias al método `forecast_states`. En este caso también se devolverán las probabilidades asociadas a cada estado.

En cuanto al entrenamiento, se han implementado dos métodos, que se pueden seleccionar mediante el enumerado `TrainingAlgorithm`. El primero permite realizar un entrenamiento supervisado, mientras que el segundo aplica el método de Baum-Welch para entrenar el modelo. Se han implementado dos formas de calcular el algoritmo de Baum-Welch: usando logaritmos o factores de escalado.

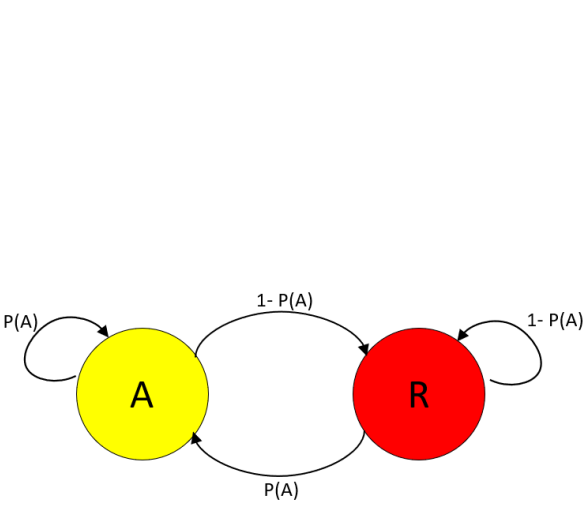
3.4. Construcción de la biblioteca

A continuación aprenderemos a construir las funcionalidades más importantes de la biblioteca: el cálculo de la verosimilitud, el entrenamiento y el cálculo de la cadena más probable (Viterbi). Para ello, volvamos nuevamente al ejemplo de nuestro amigo. En esta situación, nosotros observamos todos los días a nuestro amigo o bien con una camiseta roja o con una camiseta amarilla. Observando una secuencia de días, nos surge nuevamente la curiosidad por saber qué factores justifican la elección de camiseta de nuestro amigo. Es decir, nos preguntamos cómo identificamos el HMM más probable a la luz de la secuencia de camisetas rojas o amarillas que hemos observado.

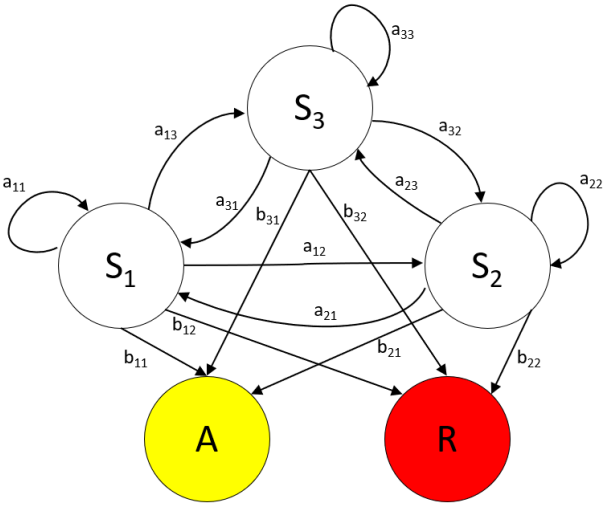
Una respuesta al ejemplo de nuestro amigo es suponer que depende de un único factor y en ese caso lo único que faltaría por determinar es cuál es la probabilidad de que se ponga una camiseta, por ejemplo, la amarilla. Podemos ver el diagrama del modelo de Markov (Figura 6a) que en este caso tiene una única variable observable.

Otra segunda respuesta podría ser que existiese una variable latente que explicase el valor de la variable observable. Este caso es el mismo que planteamos al principio de la sección 2.4, resulta que nuestro amigo podía estar feliz o triste y, en función de este estado oculto observamos un resultado con mayor probabilidad que otro. En la Figura 6c vemos como tanto la variable observable como la oculta pueden tomar dos valores y entre instantes temporales podríamos transitar entre los estados latentes. Estas probabilidades de transición, a_{ij}, aparecen en la matriz A y, las probabilidades de emisión de los valores, b_{ij}, las incluimos en otra matriz B. Para modelar este caso vamos a olvidarnos del estado inicial.

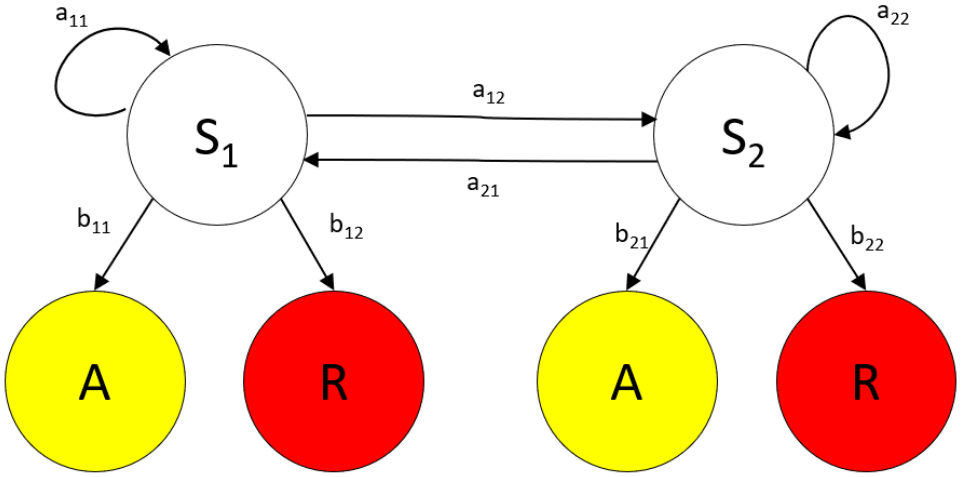
Finalmente, podríamos tener un modelo con una variable oculta y otra observable, pero la oculta toma valores en tres estados (Figura 6b).



(a) Un único factor



(b) Dos factores



(c) Tres factores

Figura 6: Tres posibles modelos de Markov que pueden explicar las observaciones de las camisetas de nuestro amigo.

Recordemos que el primer modelo tenía un único parámetro, el segundo tenía dos estados ocultos y el tercero tres. En este sentido, aquellos modelos que tengan más estados, y por tanto más parámetros que estimar, podrán "adaptarse" mejor y modelar "mejor" las observaciones, en contraposición, a los modelos pequeños les cuesta más modelar la situación. Aunque este razonamiento parezca coherente, a continuación veremos que escoger el número de estados del problema no es algo sencillo, ya que podemos incurrir en *overfitting* o *underfitting*.

En realidad, el hecho de que la calidad del ajuste sea proporcional a la cantidad de parámetros ocurre porque un modelo oculto de Markov no es más que un modelo de mixtura en el que cada estado (oculto) de la cadena tiene asociada una distribución diferente, al igual que ocurre en los modelos de mixtura. Esto permite modelar la variabilidad dentro de los datos. Al introducir nuevos estados ocultos en la cadena, estamos introduciendo nuevos grupos y, además, damos una probabilidad a estar en ese grupo y transitar desde ese grupo a otro. De esta forma, podemos aproximar los datos mediante distribuciones diferentes. Sin embargo, si introducimos muchos estados, únicamente estamos creando un

modelo "extremadamente" específico para los datos que tenemos y limitamos la capacidad de generalizar a nuevos datos. Es decir, si recibimos una nueva secuencia de observaciones e intentamos predecir (usando este modelo que hemos entrenado) los estados de las nuevas observaciones obtendremos una precisión baja. En un futuro veremos dos estrategias (AIC y BIC) para conocer el número de estados ocultos óptimo a la luz de los datos observados, que consiga el equilibrio perfecto entre ajuste y cantidad de estados.

En muchas ocasiones la estructura de nuestro modelo oculto de Markov corresponde a la casuística de nuestro problema del mundo real, esto es, que si queremos modelar si nuestro amigo está feliz o triste dependiendo del color de las camisetas, el modelo oculto de Markov tendrá dos estados ocultos dando un sentido a cada estado. Sin embargo a veces puede ayudar tener tres estados en el modelo oculto de Markov, y más tarde darle un significado a cada estado.

Ahora que tenemos clara la estructura de un modelo oculto de Markov, dado un conjunto de observaciones $\mathbf{O} = \{O_1, O_2, ..., O_T\}$ y unos parámetros del modelo $\lambda = \{A, B, \pi\}$, siendo A la matriz de transición de estados, B la matriz de emisión y π el vector de probabilidades iniciales. Nos surgen 3 cuestiones:

1. ¿Cómo podemos calcular la verosimilitud del modelo $L(\lambda|\mathbf{O})$? Es decir, dada una secuencia de observaciones, cómo de probable es que estas observaciones sean replicadas por el modelo. Esta función nos ofrece una métrica para poder elegir entre varios modelos candidatos.

2. ¿Cuál es la secuencia de estados $x_1, x_2, ..., x_T$ que dan lugar al vector de observaciones $\mathbf{O}$? Ya introducimos la importancia de esta cuestión en el apartado anterior, ya que visibilizamos la parte oculta del modelo. Hay que dejar claro que nunca sabremos cuál fue la secuencia de estados que han generado las observaciones, sin embargo escogeremos la cadena más probable.

3. ¿Cómo podemos ajustar el vector de parámetros $\lambda = \{A, B, \pi\}$ para maximizar la verosimilitud del modelo $L(\lambda \mid \mathbf{O})$? En muchas ocasiones, podemos tener un conjunto de datos en el que los estados ocultos pueden estar etiquetados o no, y vamos a utilizarlos para entrenar el HMM. Esta etapa de entrenamiento es fundamental pues nos ayuda a estimar los parámetros en base a los datos observados y así podemos emplear el HMM para hacer estimaciones de los estados ocultos con observaciones nuevas.

3.4.1. Verosimilitud del Modelo

La verosimilitud se ve como una función de los parámetros λ y se define como la probabilidad conjunta de observar unos datos dados $\mathbf{O} = \{O_1, O_2, ..., O_T\}$, si los parámetros fueran esos. Considerando también un vector de estados ocultos $\mathbf{s} = \{s_1, s_2, ..., s_T\}$, podemos escribir la verosimilitud como:

$$L(\lambda \mid \mathbf{O}) = \sum_{\mathbf{s}\in\mathbf{S}} \left(\prod_{i=1}^{n} P(O_i, \mathbf{s} \mid \lambda)\right) \quad \text{(Observaciones independientes)} \tag{3.4.1}$$

$$= \sum_{\{s_1,s_2,...,s_T\}\in\mathbf{S}} \pi_{s_1} b_{s_1}(O_1) a_{s_1,s_2} b_{s_2}(O_2) a_{s_2,s_3} \cdots a_{s_{T-1},s_T} b_{s_T}(O_T) \tag{3.4.2}$$

También podemos escribir lo anterior de forma matricial:

$$L(\lambda \mid \mathbf{O}) = \pi B(O_1)AB(O_2)A \cdots AB(O_T)1' \tag{3.4.3}$$

Donde $1'$ es un vector de unos. En ocasiones, fijaremos los parámetros del modelo y hablaremos de la verosimilitud sobre λ fijo, $P(\mathbf{O} \mid \lambda)$. Es decir, dado un modelo y una secuencia de observaciones, cómo de probable es que estas observaciones hayan sido observadas con este modelo. Esta probabilidad nos ofrece una métrica para poder elegir entre varios modelos candidatos, ya que trataremos de buscar los parámetros óptimos $\hat{\lambda}$ que maximicen la verosimilitud $P(\mathbf{O} \mid \hat{\lambda})$.

En resumen, lo que hacemos para calcular la verosimilitud es sumar todos los caminos posibles que producen esta secuencia de observaciones y, por cada camino calculamos las probabilidades de estar y observar un valor en un instante particular. La probabilidad de estar en el estado s_1 en el instante inicial viene dada por el vector π, y en ese estado la probabilidad de observar O_1 es $b_{s_1}(O_1)$. Cambiamos al instante $t = 2$ y ahora en vez de usar el vector de probabilidades iniciales, usaremos la matriz de transición, en concreto la probabilidad de transitar entre s_1 y s_2, a_{s_1,s_2} y nuevamente la probabilidad de ver esa observación $b_{s_2}(O_2)$. Repitiendo este proceso hasta llegar al instante $t = T$.

Este proceso iterativo puede ser simplificado en un esquema recursivo que hace uso de programación dinámica. En este caso tendremos un vector (α_t, con $t \in \{1, 2, \ldots, T\}$), conocido como vector de **probabilidades *forward*** que recogerá las probabilidades acumuladas hasta el instante t.

$$\alpha_1 = \pi B\left(O_1\right) \tag{3.4.4}$$

$$\alpha_t = \alpha_{t-1} AB\left(O_t\right) \quad \text{con } t \in \{1, 2, \ldots, T\} \tag{3.4.5}$$

Para implementar este algoritmo usaremos un enfoque ascendente. Para ello, guardamos las probabilidades *forward* en memoria y así calculamos fácilmente el siguiente valor. Si usásemos un enfoque descendente, tendríamos que realizar llamadas recursivas y gastaríamos más espacio en memoria guardando las cabeceras en la pila de la memoria. Esto es especialmente costoso si el vector de probabilidades *forward* es grande.

Algoritmo 1 Enfoque ascendente para calcular las probabilidades *forward*

```
Inicializar alpha[N, T]
Inicializar alpha[i, 1] = π_i b_i(O_1) para 1 ≤ i ≤ N
for t = 2 hasta T do
    for j = 1 hasta N do
        for k = 1 hasta N do
            alpha[j, t]  += alpha[k, t − 1] · a_kj · b_j(O_t)
        end for
    end for
end for
return alpha
```

Además la verosimilitud es muy fácil de calcular:

$$L(\lambda \mid \mathbf{O}) = \alpha_T 1' = \pi B(O_1)AB(O_2)A \cdots AB(O_T)1' \tag{3.4.6}$$

Complejidad del algoritmo

Veamos la ganancia que hemos conseguido al optar por un diseño basado en programación dinámica: en primer lugar, mediante el método iterativo original, el número de operaciones necesarias para calcular $L(\lambda \mid \mathbf{O})$ es del orden de $2TN^T$. Esto se debe a que en cada instante $t = 1, 2, \ldots, T$ existen N caminos posibles, consecuentemente tenemos N^T secuencias de estados posibles, y por cada estado realizamos los $2T$ cálculos de la ecuación (3.4.2). Sin embargo con el nuevo método (ecuación 3.4.6) conseguimos una complejidad del orden de TN^2 ya que en cada iteración del algoritmo calculamos las N probabilidades del vector *forward* y las conservamos en un *array*. De esta forma, haciendo uso de la memoización evitamos tener que calcular todas las rutas posibles ya que van implícitos (Figura 7) en el cálculo de la probabilidad *forward* en el instante anterior. Entonces por cada iteración simplemente tendremos que sumarlos N productos del valor de la probabilidad *forward* del instante anterior, la probabilidad de transición y la probabilidad de encontrarse en un estado concreto en ese instante.

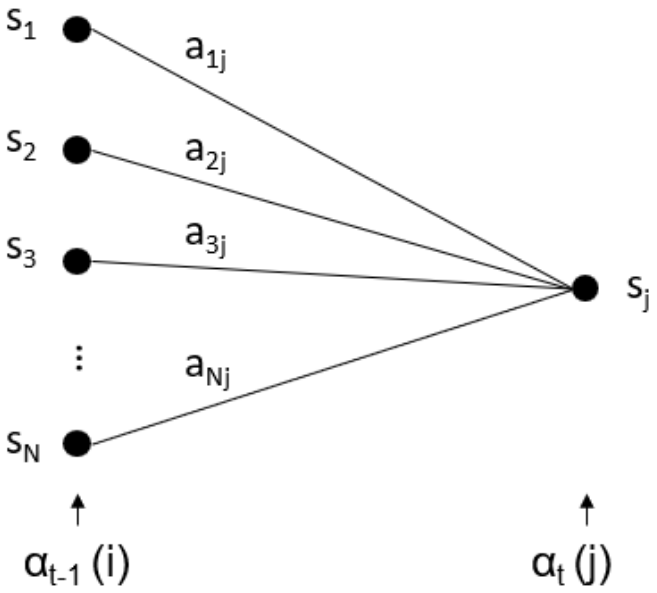


Figura 7: Diagrama de las operaciones necesarias para calcular la probabilidad *forward* $\alpha_t(j)$ en el tiempo $t = 1, 2, ..., T$ y el estado $X^{(t)} = s_j$.

Al igual que existe un procedimiento *forward*, también podemos calcular las **probabilidades *backward***. Estas probabilidades nos van a ser útiles para responder a la pregunta 3 en el proceso de optimización de los parámetros. Las portabilidades *backward* β_t es un vector que recoge las probabilidades de observar la porción restante de la secuencia de observaciones a partir de un instante t, es decir:

$$\beta_t = P(O_{t+1}, O_{t+2}, ..., O_T; X^{(t)} = s_t \mid \lambda) \tag{3.4.7}$$

Asimismo, podemos calcular de manera recursiva los valores de $\boldsymbol{\beta}$. Esta vez la recursión comienza por el final:

$$\beta_T = 1' \tag{3.4.8}$$
$$\beta_t = AB(O_{t+1})\beta_{t+1} \operatorname{con} t \in \{1, 2, ..., (T-1)\} \tag{3.4.9}$$

En el primer instante inicializamos todas las probabilidades a 1 y en el resto de iteraciones realizamos el procedimiento de cálculo de las probabilidades *backward* para cada estado s_i en un tiempo t como se ilustra en la Figura 8.

Algoritmo 2 Enfoque ascendente para calcular las probabilidades *backward*

Inicializar $beta[N, T]$
Inicializar $beta[i, T] = 1$ para $1 \leq i \leq N$
for $t = T - 1$ hasta 1 **do**
  for $j = 1$ hasta N **do**
    for $k = 1$ hasta N **do**
      $beta[j, t] \quad += beta[k, t + 1] \cdot a_{jk} \cdot b_k(O_{t+1})$
    end for
  end for
end for
return *beta*

Nótese que aunque comencemos por el instante t el enfoque usado en Algoritmo 2 sigue siendo ascendente, pues empezamos desde el caso base en el que no tenemos ninguna probabilidad *backward* calculada.

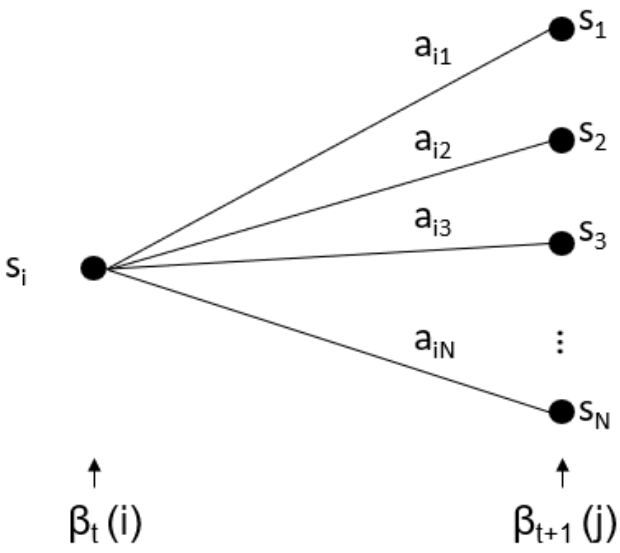


Figura 8: Diagrama de las operaciones necesarias para calcular la probabilidad *backward* $\beta_t(i)$ en el tiempo $t = 1, 2, ..., T$ y el estado $X^{(t)} = s_i$. Ahora el diagrama se lee de derecha a izquierda pues primero se calcula el valor de $\beta_{t+1}(j)$ y luego $\beta_t(i)$.

Nuevamente, el número de operaciones necesarias para calcular el vector completo $\boldsymbol{\beta}$

es del orden de TN^2 ya que seguimos agrupando en cada cálculo de $\beta_t(i)$ las probabilidades de transitar al resto de estados s_j con $j = 1, 2, ..., N$ y observar el siguiente valor O_{t+1}.

3.4.2. Decodificación

El segundo problema es encontrar la secuencia de estados "óptima" asociada a la secuencia de observación dada, esto se conoce como **decodificación** (*decoding*). Existen varias formas de afrontar el problema, pero antes hay que definir el concepto de optimalidad al construir la secuencia de estados óptima. Por ejemplo, un posible criterio de optimalidad es elegir los estados que son individualmente los más probables. Este criterio de optimalidad maximiza el número esperado de estados individuales correctos. Para ello podemos definir las **probabilidades $\boldsymbol{\gamma}$**:

$$\gamma_t(i) = P(X^{(t)} = s_i|\boldsymbol{O}, \lambda) \tag{3.4.10}$$

$\gamma_t(i)$ es la probabilidad de estar en el estado s_i en un tiempo t, dado un vector de observaciones $\mathbf{O}$ y los parámetros del modelo λ. Gracias a las probabilidades *forward* y *backward* que hemos definido antes podemos expresar $\gamma_t(i)$ como:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{L(\lambda \mid \mathbf{O})} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{3.4.11}$$

gracias a que $\alpha_t(i)$ nos da la probabilidad de estar en ese estado s_i habiendo visto todas las observaciones hasta el tiempo t, $O_1, O_2, ..., O_t$, mientras que $\beta_t(i)$ nos aporta la probabilidad de ver el resto de las observaciones $O_{t+1}, O_{t+2}, ..., O_T$. Finalmente normalizamos por la verosimilitud para que sea una probabilidad, es decir:

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{3.4.12}$$

Finalmente, podemos calcular cual es el estado s más probable en el tiempo t tomando el máximo de esta probabilidad:

$$X^{(t)} = \underset{s_i \in S}{\operatorname{argmax}} \left[\gamma_t(s_i)\right], \quad \text{con} \quad 1 \leq t \leq T \tag{3.4.13}$$

Aunque la ecuación anterior (3.4.13) maximiza el número esperado de estados correctos (eligiendo el estado más probable para cada t), esta solución trae consigo un gran problema al construir la secuencia de estados óptimos. Puede ocurrir que si el HMM tiene transiciones de estado con probabilidad cero ($a_{ij} = 0$ para algún i, j), la secuencia de estados "óptima" puede no ser válida, porque la ecuación (3.4.13) no tiene en cuenta la probabilidad de ocurrencia de los estados. Entonces si los estados s_i, s_j resultan ser los óptimos y $a_{ij} = 0$, ambos no pueden estar simultáneamente en la cadena de estados.

Para solucionar el problema anterior, una alternativa es ajustar el criterio de optimalidad. Por ejemplo, se podría buscar la secuencia de estados que maximice el número esperado de pares de estados correctos (s_t, s_{t+1}). Sin embargo, el criterio más comúnmente utilizado es encontrar el mejor camino entre los estados, es decir, maximizar $P(\mathbf{s}|\mathbf{O}, \lambda)$.

Entonces, una solución sería generar una combinación de N^T cadenas, y calcular cuál de ellas es la óptima, sin embargo la elevada complejidad computacional de este problema sugiere buscar otro enfoque. Nuevamente, existe un algoritmo de programación dinámica que nos ayuda a reducir la complejidad computacional del problema, este se conoce como algoritmo de Viterbi.

Algoritmo de Viterbi

Para encontrar la mejor secuencia de estados mediante el algoritmo de Viterbi [32], tenemos que definir las **probabilidades $\boldsymbol{\nu}$**:

$$\nu_t(j) = \max_{X^{(1)},\ldots,X^{(t-1)}} P\left(X^{(1)}\ldots X^{(t-1)}, O_1, O_2 \ldots O_t, X^{(t)} = s_j \mid \lambda\right) \tag{3.4.14}$$

Es importante destacar que en la representación del camino más probable, se toma el máximo entre todas las posibles secuencias de estados anteriores ($\max_{X^{(1)},\ldots,X^{(t-1)}}$). Al igual que otros algoritmos de programación dinámica, el algoritmo de Viterbi calcula cada valor de forma recursiva. En este sentido, una vez que se ha calculado la probabilidad de pasar por cada estado en el tiempo $t-1$, se procede a calcular la probabilidad de Viterbi tomando extendiendo los caminos más probables que llevan a la celda actual. En otras palabras, para un estado s_j dado en el tiempo t, el valor $\nu_t(i)$ se calcula de la siguiente forma:

$$\nu_t(j) = \max_{i=1}^{N} \nu_{t-1}(i) a_{ij} b_j\left(O_t\right) \tag{3.4.15}$$

Este algoritmo hace uso de dos matrices de tamaño $(N \times T)$, en una matriz guardaremos la probabilidad de cada camino y en la otra guardamos punteros al estado anterior en el camino óptimo.

Algoritmo 3 Implementación ascendente del algoritmo de Viterbi

1: $viterbi[i,1] \leftarrow \pi_i b_i(O_1)$ para $1 \leq i \leq N$
2: $puntero[i,1] \leftarrow 0$ para $1 \leq i \leq N$
3: **for** $t = 2$ hasta T **do**
4: **for** $j = 1$ hasta N **do**
5: $viterbi[j,t] = max_{i=1}^{N} \quad (viterbi[i,t-1] \cdot a_{ij}) \cdot b_j(O_t)$
6: $puntero[j,t] = argmax_{i=1}^{N} \quad (puntero[i,t-1] \cdot a_{ij}) \cdot b_j(O_t)$
7: **end for**
8: **end for**
9: Encontrar $P^* = max_{i=1}^{N} \quad viterbi[i,T]$, $z_T^* = argmax_{i=1}^{N} \quad viterbi[i,T]$
10: **for** $t = T-1$ hasta 1 **do**
11: $z_t^* = puntero[z_{t+1}^*, t+1]$
12: **end for**
13: **return** *viterbi*

En el vector $\mathbf{z}^*$ tendremos el camino óptimo que nos maximiza la probabilidad ν. Esta probabilidad se guarda en el *array viterbi*. Como vemos, el algoritmo Viterbi es

idéntico al algoritmo para calcular las probabilidades *forward* (3.4.4, 3.4.5) excepto que toma el camino que maximiza las probabilidades de los caminos creados hasta ese instante, mientras que el algoritmo *forward* toma la suma.

Complejidad del algoritmo

Como podemos ver, la complejidad de este algoritmo es $O(TN^2)$. Con la intención de lograr comprender en profundidad el algoritmo, veamos en la Figura 9 cómo al tomar el máximo en cada iteración estamos podando el árbol que contiene los N^T caminos posibles. En el siguiente ejemplo (Figura 9), queremos determinar la secuencia de estados de frío (C) o calor (H) de los últimos tres días basándonos en la cantidad de helados vendidos. La temperatura será la variable latente y la cantidad de helados será la variable observada.

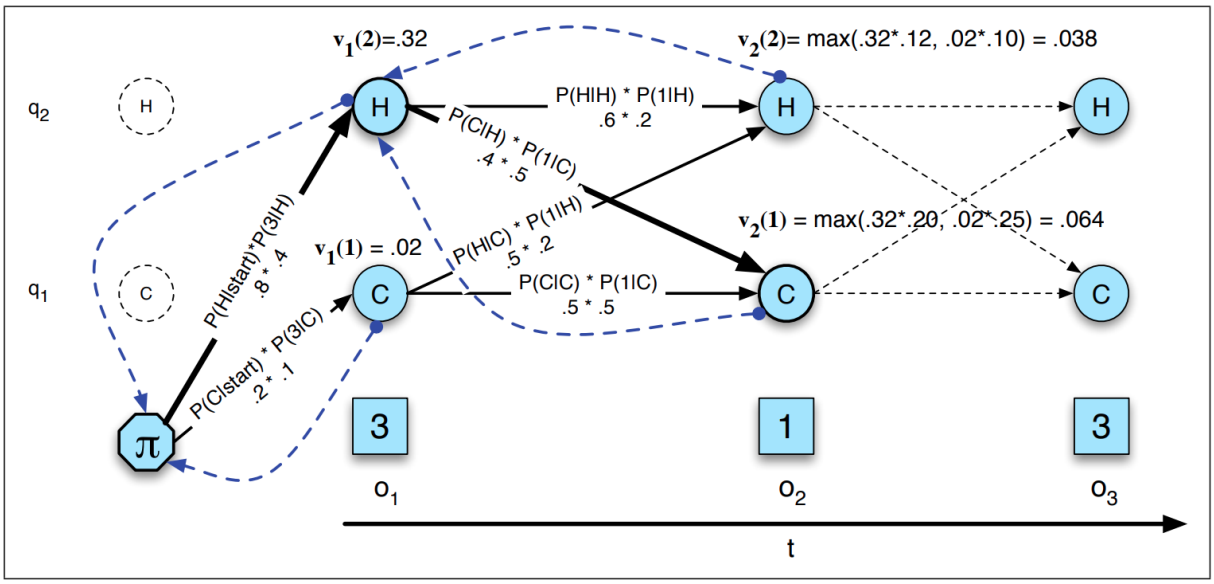


Figura 9: Diagrama de algoritmo de Viterbi[13] que muestra en negrita el camino más probable hasta el momento y los punteros de cada camino (líneas azul discontinuas).

En este caso, si nos situamos en el instante $t = 2$, la variable $X^{(2)}$ puede tomar dos valores H o C. De esta forma, el algoritmo comenzará evaluando la posibilidad $X^{(2)} = H$, en el paso previo, $X^{(1)}$ podría ser otro estado H con una probabilidad total de $0{,}32 \cdot 0{,}12 = 0{,}038$, o tomar el valor C que tiene una probabilidad de $0{,}02 \cdot 0{,}10 = 0{,}002$. Ahora, para construir el siguiente camino posible hacia $X^{(3)}$, únicamente vamos a considerar el camino cuya secuencia de estados ha sido (H, H), ya que los caminos potenciales (H, H, H) y (H, H, C) van a tener siempre más probabilidad que (C, H, H) o (C, H, C). Finalmente, quedaría evaluar la posibilidad de $X^{(2)} = C$, en ese caso la secuencia más probable es (H, C) con una probabilidad de $0{,}32 \cdot 0{,}20 = 0{,}64$. En general, cuando llegamos a un estado $X^{(t)}$ en el instante $t > 1$, siempre escogemos el camino con probabilidad máxima y descartamos el resto de caminos, ya que al ampliar el camino con el siguiente estado, la probabilidad resultante de usar cualquier otro camino que pase por el estado $X^{(t)}$ seleccionado será necesariamente menor.

3.4.3. Entrenamiento del Modelo Oculto de Markov

Dada una secuencia de observaciones $\mathbf{O} = \{O_1, O_2, ..., O_T\}$, podemos conseguir que el HMM aprenda los parámetros A, B y π que mejor ajustan el modelo a la secuencia, esto es, que maximizan la probabilidad de observar la secuencia $\mathbf{O}$ dado el modelo $\lambda = \{A, B, \pi\}$. Existen varias formas de estimar los parámetros λ del modelo, las dos más comunes son maximización directa o maximización de las expectativas (*Expectation-Maximization*). En este trabajo usaremos el método de Baum-Welch [2] que es un caso particular del algoritmo de maximización de las expectativas.

Tanto el algoritmo de maximización directa como el de Baum-Welch, calculan iterativamente probabilidades y, por lo tanto, la multiplicación de números positivos menores que 1 resulta en *underflow* en unas pocas iteraciones. Además, ambos algoritmos tienen como fin último maximizar la función de verosimilitud variando los parámetros λ del modelo. Sin embargo, nos podemos encontrar con la existencia de múltiples máximos locales en la función de verosimilitud y, pese a que el objetivo sea encontrar el máximo global, a menudo no lograremos encontrarlo ya que esta búsqueda depende de los parámetros iniciales en la optimización. Por lo que usaremos un rango de valores de partida para la maximización, y comprobaremos si llegamos al mismo máximo en cada caso. En los siguientes apartados, discutiremos algunas formas de superar estos impedimentos.

Expectation Maximization

El algoritmo EM es el algoritmo más utilizado a la hora de buscar los parámetros que maximicen la función de verosimilitud. Pese a que hay varios autores como Leroux y Puterman [14] o Robert y Titterington [23] que hacen una comparativa entre distintos métodos de optimización, Zucchini, MacDonald y Langrock [36] critican que pese a que la comunidad tiende a preferir este algoritmo por su facilidad de programación y evitar el cálculo de derivadas, la realidad es que en el caso particular de los HMMs aplicar EM puede ser más costoso ya que hay que calcular una probabilidad *backward* y una *forward*, frente a una simple *forward* en el caso de maximización directa.

El algoritmo EM busca optimizar la función de verosimilitud en los casos en que existe un modelo que depende probabilísticamente de una componente observada y otra latente. Los HMM tienen tanto un componente no observado, los estados ocultos, como las observaciones reales, haciendo de los HMM una clase de modelos para los que el algoritmo EM puede ser aplicable. En general, el algoritmo EM es un método iterativo para estimar el máximo verosímil, y aprovecha que la log-verosimilitud de los datos completos (*Complete Data Log-Likelihood*, CDLL) es fácil de maximizar incluso si la verosimilitud de los datos observados no lo es. La CDLL es la verosimilitud logarítmica de los parámetros de interés λ, basada tanto en los datos observados como en los que no. El algoritmo fue propuesto por Little y Rubin en 2002 [16] y consta de dos partes:

1. **Paso E** en el que se calculan las expectativas de los datos no observados dadas unas observaciones y las estimaciones de los parámetros λ hasta el momento.

2. **Paso M** en el que se realiza una maximización de la CDLL respecto a los parámetros.

Cuando hablamos de los "datos no observados", nos referimos a una función de los datos no observados que aparecerá en el cálculo de la CDLL, como dicen Little y Rubin [16] esta es la idea principal del algoritmo EM.

Baum-Welch

El algoritmo de Baum-Welch es un caso particular del algoritmo EM específico para los modelos ocultos de Markov. Como el objetivo principal es maximizar la CDLL, haremos uso de las probabilidades *forward* (3.4.6) y *backward* (3.4.19). Adicionalmente, haremos uso de las probabilidades $\boldsymbol{\gamma}$ definidas en (3.4.10) y también tendremos que definir las **probabilidades $\boldsymbol{\xi}$**:

$$\xi_t(i,j) = P(X^{(t)} = s_i, X^{(t+1)} = s_j | \mathbf{O}, \lambda) \tag{3.4.16}$$

$\xi_t(i,j)$ es la probabilidad de estar en el estado s_i en el instante t y en el instante $t+1$ estar en el estado s_j. Podemos expresar gráficamente esta probabilidad (Figura 10):

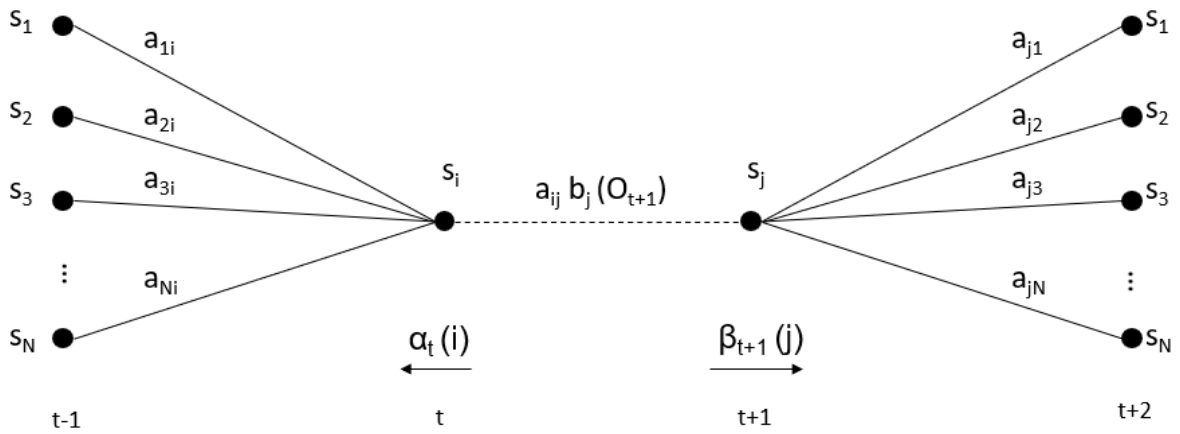


Figura 10: Diagrama que ilustra las operaciones necesarias para calcular la probabilidad de que el sistema se encuentre en un estado s_i en el instante t y en el estado s_j en el instante $t+1$

A raíz de la imagen anterior, vemos cómo se puede escribir esta probabilidad en función de las probabilidades *forward* y *backward*:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \beta_{t+1}(j)}{L(\lambda \mid \mathbf{O})} \tag{3.4.17}$$

$$= \frac{\alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \beta_{t+1}(j)} \tag{3.4.18}$$

a modo de aclaración, recordemos que la verosimilitud la podíamos escribir en función de las probabilidades *forward* (3.4.6), también podemos escribir la verosimilitud en función de las probabilidades *backward*:

$$L(\lambda \mid \mathbf{O}) = \sum_{i=1}^{N} \pi \cdot \boldsymbol{\beta}_0 \tag{3.4.19}$$

Y también podemos expresar la verosimilitud en función de las dos probabilidades *forward* y *backward* como nos ilustra la Figura (10):

$$L(\lambda \mid \mathbf{O}) = \boldsymbol{\alpha}_t \boldsymbol{\beta}_t' \quad \text{para cualquier } t = 1, 2, \ldots, T. \tag{3.4.20}$$

Por ello, el denominador de la ecuación (3.4.18) únicamente está haciendo uso de la ecuación anterior (3.4.20) escrito de forma expandida en lugar de forma matricial. Este denominador $L(\lambda \mid \mathbf{O})$ sirve para normalizar $\boldsymbol{\xi}$ y que tenga medida 1.

Asimismo podemos relacionar las probabilidades $\boldsymbol{\gamma}$ y $\boldsymbol{\xi}$ de la siguiente forma:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j) \tag{3.4.21}$$

Si sumamos los $\gamma_t(i)$ durante todo el tiempo t ($\sum_{t=1}^{T} \gamma_t(i)$) obtenemos la cantidad esperada de veces que se ha visitado el estado s_i. De igual forma, si sumamos los $\xi_t(i, j)$ durante todo el tiempo t ($\sum_{t=1}^{T} \xi_t(i, j)$) obtenemos la cantidad esperada de veces del estado s_i al estado s_j. Estos dos conceptos nos van a ser útiles a la hora de reestimar los parámetros λ del modelo oculto de Markov.

Las etapas del algoritmo EM serán entonces las siguientes:

1. **Paso E**: Por cada iteración del algoritmo calcularemos las estimaciones de las probabilidades $\boldsymbol{\gamma}$ y $\boldsymbol{\xi}$ dado el vector de observaciones $\mathbf{O}$ y los parámetros ajustados de la iteración anterior λ (en la primera iteración usaremos unos parámetros inicializados[2])

2. **Paso M**: Una vez que hemos calculado los estimadores para $\boldsymbol{\gamma}$ y $\boldsymbol{\xi}$, tenemos que maximizar la CDLL respecto a nuestros parámetros $\lambda = \{A, B, \pi\}$. Esto implica encontrar los parámetros $\bar{\pi}$, $\bar{A}$ y $\bar{B}$. De forma que:

   a) $\bar{\pi}_i$ sea el número de veces que nos encontramos en el estado s_i en el instante inicial, que equivale a $\gamma_1(i)$

   b) $\bar{a}_{ij}$ sea el número de veces que transitamos del estado s_i a s_j, entre el número total de veces que transitamos desde el estado s_i,

$$\frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

[2]Existen varias estrategias a la hora de escoger los valores iniciales de estos parámetros, en *Hidden Markov Models for Time Series*[36] aconsejan usar estrategias basadas en cuantiles de las observaciones. Por ejemplo, si el HMM tiene tres estados ocultos usar el primer cuartil, la mediana y el tercer cuartil de las observaciones. Por otro lado recomienda inicializar la matriz A dando las mismas probabilidades de transitar entre estados, es decir, que $a_{ij} = c$ con $0 \leq c \leq 1$ para $i \neq j$ y el resto de valores ajustarlo para que las filas sumen 1.

c) $\bar{b}_i(O)$ sea el número de veces que estamos en el estado s_i y vemos la observación O entre el número de veces que estamos en el estado s_j. Esta última reestimación depende directamente del problema que se esté modelando, es decir, que si se trata de un HMM de Poisson tendremos unas funciones de distribución que modelen las observaciones y será distinto de si tenemos un HMM normal. En resumen $\bar{b}_i(O)$ será:

$$\frac{\sum_{\substack{t=1 \\ \text{s.a. } O}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Según Baum et al.[2], si queremos maximizar la CDLL tenemos que maximizar las siguientes ecuaciones:

a) $\sum_{j=1}^{N} \gamma_j(1) \log \pi_j$ para obtener π.

b) $\sum_{j=1}^{N} \sum_{k=1}^{N} \left(\sum_{t=2}^{T} \xi_{jk}(t) \right) \log \gamma_{jk}$ para obtener A

c) $\sum_{j=1}^{N} \sum_{t=1}^{T} \gamma_j(t) \log P\left(O_t | X^{(t)} = s_j\right)$ para obtener los parámetros de las distribuciones de emisión.

Los resultados de las optimizaciones son las siguientes:

a) $\pi_j = \gamma_j(1)$.

b)

$$jk = \sum_{t=2}^{T} \xi_{jk}(t) / \sum_{k=1}^{N} \sum_{t=2}^{T} \xi_{jk}(t)$$

c) La maximización de los parámetros de las distribuciones de emisión depende de cada caso particular. En el artículo original de Baum et al. [2] mencionan el ejemplo de una distribución normal(μ, σ). Donde los resultados de las optimizaciones de los parámetros son:

$$\widehat{\mu}_j = \frac{\sum_{t=1}^{T} \gamma_t(j) O_t}{\sum_{t=1}^{T} \gamma_t(j)}$$

y

$$\widehat{\sigma}_j^2 = \frac{\sum_{t=1}^{T} \gamma_t(j) \left(O_t - \widehat{\mu}_j\right)^2}{\sum_{t=1}^{T} \gamma_t(j)}$$

Aunque Baum et al. [2] menciona cómo resolver las optimizaciones del "paso M", múltiples autores [21] demuestran los resultados obtenidos usando Multiplicadores de Lagrange.

Como ya hemos mencionado existen varias trabas a la hora de llevar los HMM a la práctica, entre ellas se encuentra el desbordamiento. Como la verosimilitud está compuesta de un producto de probabilidades, es necesario realizar un escalado para evitar que esta sea cada vez más pequeña y evitar que tienda a 0. Para evitar esto existen dos métodos: usar logaritmos o factores de escalado para escalar todos los vectores de probabilidad (*forward*, *backward*, γ, ξ).

Comencemos discutiendo las ventajas de ambos. En primer lugar, aunque el enfoque logarítmico es más rápido y sencillo que el de factores de escalado. No obstante, es más usado el segundo, especialmente en modelos espacio-estados genéricos (por ejemplo, el algoritmo de Filtrado de Kalman usa esta misma idea). La principal fortaleza del algoritmo es tratar con un mayor conjunto de datos. Esto se debe a que en el enfoque logarítmico a veces es necesario calcular la probabilidad sin escalar para realizar sumas (por ejemplo, al realizar una multiplicación matricial), consecuentemente nos podremos encontrar problemas de overflow.

Veamos ahora cómo realizar el escalado usando factores de escalado. La idea es transformar el vector de probabilidades *forward* de tal forma que en cada iteración del algoritmo, los pesos de $\boldsymbol{\alpha}_t$ sumen 1:

$$\sum_{i=1}^{N} \alpha_t(i) = 1, \qquad \text{para cualquier } t = 1, 2, \ldots, T \tag{3.4.22}$$

Para diferenciar las variables escaladas vamos a definir para cada $t = 1, 2, \ldots, T$, el vector:

$$\hat{\alpha}_t = \frac{\alpha_t}{w_t} = C_t \cdot \alpha_t \tag{3.4.23}$$

donde $w_t = \sum_{i=1}^{N} \alpha_t(i)$ es el peso por el que dividimos para normalizar la suma y cumplir con (3.4.22) y, $C_t = 1/w_t$. Vamos a volver a expresar el vector de probabilidades *forward* con los nuevos reescalados, para simplificar las cosas vamos a definir $\bar{\alpha}_t$ que corresponde al valor antes de ser escalado. Por lo que escribiremos el siguiente problema recursivo:

$$\begin{aligned}
\bar{\alpha}_1(i) &= \alpha_1(i) \\
\hat{\alpha}_1 &= \frac{\bar{\alpha}_1(i)}{\sum_i \bar{\alpha}_1(i)} = \hat{c}_1 \cdot \bar{\alpha}_1(i) \\
\bar{\alpha}_{t+1}(j) &= \sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j \left(O_{t+1}\right) \\
\hat{\alpha}_{t+1}(i) &= \frac{\bar{\alpha}_{t+1}(i)}{\sum_i \bar{\alpha}_{t+1}(i)} = \hat{c}_{t+1} \bar{\alpha}_{t+1}(i)
\end{aligned} \tag{3.4.24}$$

En el anexo (sección 6.1) podemos encontrar una demostración constructiva de cómo se cumple la ecuación (3.4.23).

Podemos reescribir la verosimilitud $L(\lambda \mid \mathbf{O})$ como:

$$L(\lambda \mid \mathbf{O}) = \sum_{i=1}^{N} \alpha_T(i) = \frac{1}{C_T} = w_T \tag{3.4.25}$$

Nótese que aunque hayamos solucionado el problema de *underflow* en las probabilidades *forward*, únicamente hemos trasladado el problema a los factores de escalado C_T

y consecuentemente tenemos un problema de *overflow* en el cálculo de la verosimilitud. Nótese que este problema ocurre porque nuevamente para calcular C_T nos ayudamos del vector $\hat{c}_\tau$ cuyos valores son menores que 1. Entonces, puesto que $C_T = \prod_{\tau=1}^{T} \hat{c}_\tau$, obtendremos *underflow* al calcular C_T. Luego, para calcular la verosimilitud sin preocuparnos del desbordamiento tomaremos logaritmos en los cálculos:

$$\begin{aligned}
\log L(\lambda \mid \mathbf{O}) &= \log(\frac{1}{C_T}) \\
&= \log(\frac{1}{\prod_{\tau=1}^{T} \hat{c}_\tau}) \\
&= -\log(\prod_{\tau=1}^{T} \hat{c}_\tau) \\
&= -\sum_{\tau=1}^{T} \log(\hat{c}_\tau)
\end{aligned}$$

Por otro lado, para el escalado de las probabilidades *backward* usaremos implícitamente las mismas variables de escalado C_t y $\hat{c}_t$ que hemos definido al escalar las probabilidades *forward* (3.4.23, 6.1.1). Asimismo, definiremos la variable D_t para escalar las variables $\boldsymbol{\beta}$ y conseguir:

$$\hat{\beta}_t(i) = D_t \beta_t(i) \tag{3.4.26}$$

El término D_t lo definiremos de la siguiente forma:

$$D_t = \prod_{\tau=t}^{T} c_\tau \tag{3.4.27}$$

También podemos relacionar D_t con C_t de la siguiente forma:

$$C_t D_{t+1} = \prod_{\tau=1}^{t} c_\tau \prod_{\tau=t+1}^{T} c_\tau = \prod_{\tau=1}^{T} c_\tau = C_T \tag{3.4.28}$$

Gracias a la anterior definición de D_t (3.4.27), conseguimos un método recursivo de calcular las probabilidades *backward* idéntico al de las probabilidades *forward* (6.1.1).

$$\begin{aligned}
\bar{\beta}_T(i) &= \beta_T(i) \\
\bar{\beta}_t(j) &= \sum_{i=1}^{N} a_{ij} b_j\left(O_{t+1}\right) \hat{\beta}_{t+1}(i) \\
\hat{\beta}_t(i) &= c_t \bar{\beta}_t(i)
\end{aligned} \tag{3.4.29}$$

Nótese que con la definición anterior de $\hat{\beta}_t(i)$ (3.4.26) no estamos imponiendo la condición

$$\hat{\beta}_t(i) = \frac{\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)} \tag{3.4.30}$$

ya que esto último no es cierto pues $C_t \neq D_t$.

Una vez que tenemos $\hat{\boldsymbol{\alpha}}$ y $\hat{\boldsymbol{\beta}}$ podemos calcular unos nuevos valores de $\boldsymbol{\gamma}$ y $\boldsymbol{\xi}$ usando (3.4.23) y (3.4.26):

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \beta_{t+1}(j)}{L(\lambda \mid \mathbf{O})} \tag{3.4.31}$$

$$= \hat{\alpha}_t(i) a_{ij} b_j\left(O_{t+1}\right) \hat{\beta}_{t+1}(j) \frac{1}{L(\lambda \mid \mathbf{O})} \frac{1}{C_t} \frac{1}{D_{t+1}} \tag{3.4.32}$$

Y usando $C_t D_{t+1} = C_T$ (3.4.28) y que $L(\lambda \mid \mathbf{O}) = \sum_{i=1}^{N} \alpha_T(i)$ (3.4.6),

$$L(\lambda \mid \mathbf{O}) = \sum_{i=1}^{N} \alpha_T(i) = w_T = \frac{1}{C_T}$$
$$L(\lambda \mid \mathbf{O}) \cdot C_T = 1 \tag{3.4.33}$$

Por lo que la ecuación 3.4.32 se reescribe como:

$$\begin{aligned} \xi_t(i,j) &= \hat{\alpha}_t(i) a_{ij} b_j\left(O_{t+1}\right) \hat{\beta}_{t+1}(j) \frac{1}{L(\lambda \mid \mathbf{O})} \frac{1}{C_t} \frac{1}{D_{t+1}} \\ &= \hat{\alpha}_t(i) a_{ij} b_j\left(O_{t+1}\right) \hat{\beta}_{t+1}(j) \frac{1}{L(\lambda \mid \mathbf{O}) \cdot C_T} && \text{(usando (3.4.28))} \\ &= \hat{\alpha}_t(i) a_{ij} b_j\left(O_{t+1}\right) \hat{\beta}_{t+1}(j) && \text{(usando (3.4.33))} \end{aligned} \tag{3.4.34}$$

Y ahora usamos la nueva variable $\boldsymbol{\xi}$ para calcular $\boldsymbol{\gamma}$:

$$\begin{aligned} \gamma_t(i) &= \sum_{j=1}^{N} \xi_t(i,j) \\ &= \frac{1}{L(\lambda \mid \mathbf{O})} \alpha_t(i) \beta_t(i) \\ &= \hat{\alpha}_t(i) \hat{\beta}_t(i) \frac{1}{L(\lambda \mid \mathbf{O})} \frac{1}{C_t} \frac{1}{D_t} \\ &= \hat{\alpha}_t(i) \hat{\beta}_t(i) \frac{1}{c_t} \end{aligned} \tag{3.4.35}$$

Finalmente, el algoritmo de Baum-Welch, independientemente del tipo de escalado, se resume en:

Algoritmo 4 Algoritmo de Baum-Welch

Inicializar la verosimilitud $L \leftarrow -\infty$
for $t = 1$ hasta $max\_iteraciones$ **do**
  $alfa \leftarrow$ Matriz probabilidades *forward* según ecuación (3.4.4)
  $beta \leftarrow$ Matriz probabilidades *backward* según ecuación (3.4.8)
  $xi \leftarrow$ Matriz probabilidades ξ según ecuación (3.4.18)
  $gamma \leftarrow$ Matriz probabilidades γ según ecuación (3.4.10)
  $pi[i] \leftarrow gamma[i, 1]$ para $1 \leq i \leq N$
  for $j = 1$ hasta N **do**
    for $k = 1$ hasta N **do**
      Inicializamos la variable auxiliar $denom = 0$
      for $t = 1$ hasta $T - 1$ **do**
        $a[j, i]$+= $xi[j, t, i]$
        $denom$ += $gamma[j, t]$
      end for
      if $denom == 0$ **then**:
        $a[j, i] \leftarrow 0$ **else**
        $a[j, i]$ /= $denom$
      end if
    end for
  end for
  for $i = 1$ hasta N **do**
    Inicializamos $num \leftarrow 0$ y $den \leftarrow 0$
    for $t = 1$ hasta $T - 1$ **do**
      num += $(gamma[i, t] \times O[t])$
      den += $gamma[i, t]$
    end for
    $mu[i] \leftarrow \frac{num}{den}$
  end for
  for $i = 1$ hasta N **do**
    Volvemos a inicializar los valores $num \leftarrow 0$ y $den \leftarrow 0$
    for $t = 1$ hasta $T - 1$ **do**
      num += $gamma[i, t] \times ((O[t] - mu[i])^2)$
      den += $gamma[i, t]$
    end for
    $sigma[i] \leftarrow \sqrt{\frac{num}{den}}$
  end for
  Calculamos un nuevo valor de verosimilitud, L'
  if $L' - L < precision$ **then**:
    return L'
  end if
  $L \leftarrow L'$
end for
return L

Entrenamiento supervisado

El entrenamiento supervisado consistirá en estimar la distribución inicial, las distribuciones de transición y los parámetros de la distribución de emisión. Para ilustrar cómo se realiza esta estimación, supondremos que tendremos una variable discreta llamada `LABEL` que pertenece al conjunto de datos etiquetados y nos indica el estado del modelo oculto de Markov en un tiempo t.

Para estimar la distribución inicial $\hat{\pi}$ haremos la siguiente proporción:

$$\pi_i = \frac{[\texttt{LABEL} = i]}{\sum_{j=1}^{N} \texttt{LABEL} = j} = \frac{[\texttt{LABEL} = i]}{T}, \text{ con } 1 \leq i \leq N \tag{3.4.36}$$

Es decir, estimaremos $\hat{\pi}$ usando la proporción de estados que pasamos en el estado i entre todos los tiempos. Asimismo, estimaremos las probabilidades de transición de estado como:

$$\hat{A} = \begin{pmatrix} P([\texttt{LABEL}_{t+1} = 0]|[\texttt{LABEL}_t = 0]) & \cdots & P([\texttt{LABEL}_{t+1} = N]|[\texttt{LABEL}_t = 0]) \\ P([\texttt{LABEL}_{t+1} = 0]|[\texttt{LABEL}_t = 1]) & \cdots & P([\texttt{LABEL}_{t+1} = N]|[\texttt{LABEL}_t = 1]) \\ \vdots & & \vdots \\ P([\texttt{LABEL}_{t+1} = 0]|[\texttt{LABEL}_t = N]) & \cdots & P([\texttt{LABEL}_{t+1} = N]|[\texttt{LABEL}_t = N]) \end{pmatrix} \tag{3.4.37}$$

La estimación de la matriz de transición $\hat{A}$ se calculará mediante las probabilidades de transitar al estado i, estando en el estado j en el instante anterior, para cualquier par de valores $1 \leq i, j \leq N$. Más concretamente:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} I_i(t) * I_j(t+1)}{\sum_{t=1}^{T-1} (I_i(t) * I_k(t+1))} \tag{3.4.38}$$

con $k = 1, 2, ..., N$ donde $I_i(t)$ es la indicatriz del estado i en el tiempo t. Es decir:

$$I_i(t) == \begin{cases} 1 & \text{si } \texttt{LABEL}_t = i \\ 0 & \text{si } \texttt{LABEL}_t \neq i \end{cases}$$

Nótese que llegamos hasta el estado $T-1$ porque desde el estado x_T no transitamos a ningún otro estado. En lenguaje natural, estamos contando las veces que transitamos del estado x_i al estado x_j entre todas las posibles transiciones desde el estado x_i.

Finalmente, para estimar los parámetros de las distribuciones de emisión usaremos estadísticos suficientes. Un estadístico suficiente es una función de los datos que expresa toda la información sobre el parámetro θ que estamos estudiando. En nuestro caso particular $\theta = \{\mu, \sigma\}$ de cada distribución de emisión, es decir, por cada estado tendremos unos parámetros θ correspondiente a la distribución de emisión.

Para una distribución normal con media y varianza desconocidas, los estadísticos suficientes son:

$$\hat{\mu} = \frac{\sum_{i=1}^{N} O_i}{N} \tag{3.4.39}$$

$$\hat{\sigma} = \frac{1}{N} \sum_{i=1}^{N} (O_i - \mu)^2 \tag{3.4.40}$$

Gracias a estos estadísticos podemos estimar los parámetros, sin necesidad de conocer la probabilidad de estar en un estado en particular dado una observación específica.

3.4.4. División y evaluación del conjunto de datos.

Existen dos objetivos principales respecto al uso de un modelo de *machine learning*: ajustarlo a un conjunto de observaciones y realizar inferencia. A veces se tiene un único conjunto de datos para entrenar y evaluar el modelo, sin embargo, no sería correcto usar el mismo conjunto de datos para ambas tareas. Esto es porque el modelo se ha sobreajustado a los datos usados para el entrenamiento, entonces cuando el modelo intenta estimar los estados ocultos usando los datos que ya ha visto lo hace con una mayor exactitud y las métricas que obtenemos son completamente ilusorias.

Es por esto que el método `train` también ofrece la opción de realizar una división del conjunto de datos en un conjunto de entrenamiento y otro de evaluación. Esta idea está pensada fundamentalmente para el entrenamiento supervisado, ya que dispone de un conjunto de datos etiquetados contra los que podremos comparar las predicciones del modelo. Asimismo, existe la posibilidad de entrenar el modelo mediante un algoritmo no supervisado, pero añadir este conjunto de datos etiquetados para evaluar el modelo. En ese caso, sí nos interesará poder dividir el conjunto de datos principal entre entrenamiento y evaluación.

Aunque el tamaño de la división suele depender del problema en cuestión, la división suele ser 80 % para el conjunto de entrenamiento y 20 % para el conjunto de prueba. No obstante, es importante notar que esta división del conjunto de datos no se debe hacer bruscamente, partiendo el conjunto de datos por el percentil ochenta. De esta forma podríamos obtener valores en las métricas de evaluación que no se corresponden con el valor real de la población. Para resolver este problema se aplica una técnica llamada ***cross-validation***.

Cross-validation consiste en dividir el conjunto de datos en múltiples subconjuntos, conocidos como "pliegues" (*fold*). El algoritmo recibe el nombre de *k-fold cross-validation* ya que divide el conjunto de datos en k pliegues[3] de aproximadamente igual tamaño. Luego, se realiza el entrenamiento y la evaluación del modelo k veces, una por cada pliegue. Entonces en cada iteración se selecciona un pliegue que será usado para la evaluación y el resto de subconjuntos se usarán para el entrenamiento. De esta forma, calcularemos el valor de la métrica real usando una media ponderada de los valores de cada evaluación evaluados en cada etapa del algoritmo. Así aunque el entrenamiento y la evaluación

[3]Se suele usar $k = 5$ o $k = 10$, no obstante, depende del problema que se esté tratando. También existe un caso extremo con $k = N$ siendo N el número de registros del conjunto de datos. Este sistema *leave-one-out*, es muy costoso computacionalmente pero ayuda a valorar cómo se comporta el modelo ante un único caso.

del modelo sea k veces más complejo, se obtendrá una selección más precisa de los hiperparámetros del modelo, ya que este algoritmo permite evaluar cómo se generaliza el modelo a diferentes subconjuntos de los datos.

Existen dos ventajas claras de este procedimiento: en primer lugar cada pliegue es sometido a prueba por lo que el resultado de la métrica realmente recoge cada una de las entradas del conjunto de datos, mientras que la división anterior por el percentil ochenta, únicamente recogía el veinte porciento restante. Por otro lado, reducimos la varianza del resultado ya que, sobretodo en conjuntos de datos pequeños (menos de mil registros), al dividir el conjunto de datos y apartar un subconjunto para pruebas, estamos retirando parte de información sobre la población que luego se reflejará directamente en la etapa de evaluación del modelo. Asimismo, si usamos todos los datos para entrenamiento reduciremos la varianza de la métrica ya que siempre incluiremos todos los registros del conjunto de datos principal.

Hay que recalcar, que existen situaciones en las que los datos están correlacionados. Por ejemplo, en el problema que nos ocupa (sección 4) tratamos con una serie temporal donde no podemos hacer esta división aleatoria, ya que los datos presentan un orden temporal. Por ello, antes de dividir los datos y aplicar *cross-validation* hay que cerciorarse que se incluye el orden de los datos en la división, así no evaluamos un modelo usando datos de instantes temporales anteriores a los que se han usado para entrenarlo. Para abordar este problema, existen varias técnicas de validación cruzada específicas para series temporales, entre las que destacan: la validación cruzada progresiva y la validación cruzada en bloque.

La validación cruzada progresiva consiste en dividir el conjunto de datos principal en bloques de entrenamiento y prueba de manera secuencial. En cada iteración, se utiliza una ventana temporal para definir el conjunto de entrenamiento y el conjunto de prueba a continuación. De esta manera, se garantiza que el conjunto de prueba siempre está en el futuro del conjunto de entrenamiento.

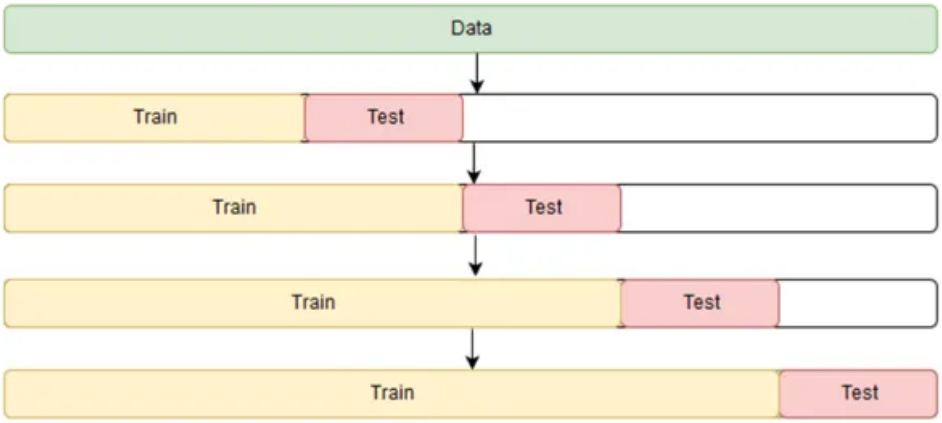


Figura 11: [26] Imagen que ilustra el proceso de validación cruzada progresiva.

Sin embargo, esto puede introducir información en el modelo que ya se ha visto, por lo tanto el modelo los memorizará. Para evitar esto, se introdujo la validación cruzada en bloque. Funciona cogiendo bloques de igual tamaño a lo largo del tiempo, cerciorándose que el subconjunto de prueba queda en una ventana temporal posterior al conjunto de

entrenamiento. Para que trabajemos con todos los datos, la siguiente tanda empezará en el conjunto de prueba anterior.

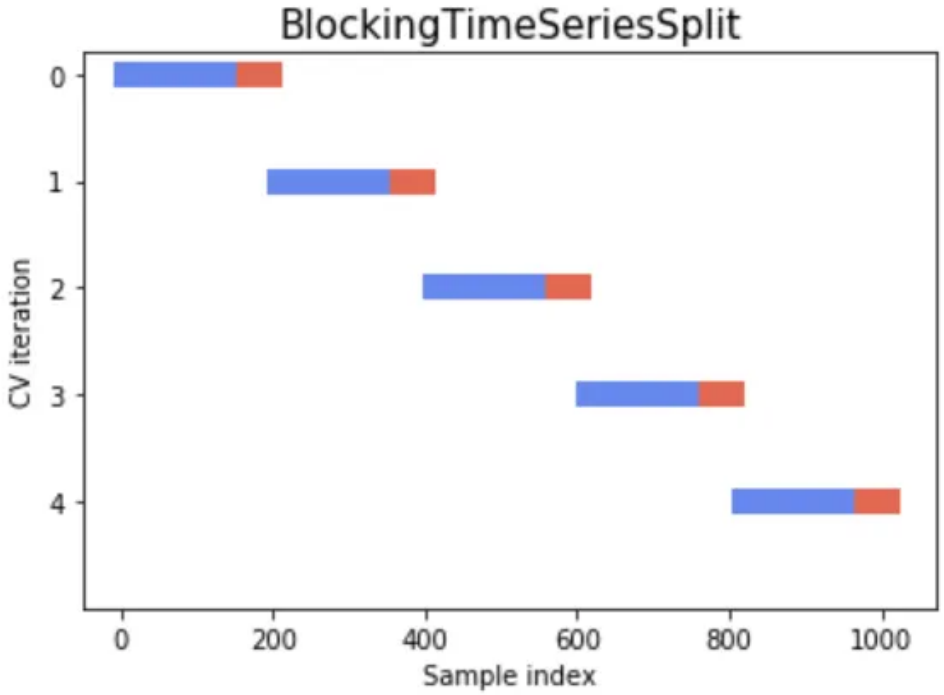


Figura 12: [26] Imagen que ilustra el proceso de validación cruzada en bloque.

Una vez explicado el proceso de división de los datos, podemos proceder a comprender cómo debemos evaluar las predicciones. Para ello hay que introducir nuevos conceptos como matriz de confusión, funciones de pérdida y métricas de evaluación.

En primer lugar, cuando tratamos con modelos de predicción binaria, como en el caso que trataremos al final del trabajo, podemos distinguir cuatro resultados: Un falso positivo (FP), que indica que se predice la existencia de un suceso cuando en realidad no lo hay, un verdadero positivo (TP), que indica que se predice correctamente un suceso, asimismo, podemos tener un falso negativo (FN), que indica que se predice que no ocurre un suceso cuando en realidad sí y un verdadero negativo (TN), que indica que se predice correctamente que no ocurre un suceso.

		Predicciones	
	Población total = P + N	Positivos (PP)	Negativos (PN)
Estados Actuales	Positivos (P)	Verdadero Positivo (TP)	Falso Negativo (FN)
	Negativos (N)	Falso Positivo (FP)	Verdadero Negativo (TN)

Tabla 3: Matriz de confusión con los posibles tipos de resultados obtenidos en una clasificación binaria.

Estos conceptos se recogen bajo el nombre de test de diagnóstico ya que la predicción se puede interpretar como un diagnóstico de la situación. Existen dos casos en los que fallamos al hacer un diagnóstico, los falsos positivos (FP) y los falsos negativos (FN). Dependiendo del problema real que estemos modelando nos interesa minimizar un error

u otro. Por ejemplo, en el caso de un modelo que trata de predecir enfermedades, es preferible minimizar los falsos negativos para asegurar que un mayor número de personas sean diagnosticadas y reciban tratamiento. Por otro lado, en el caso de un modelo que predice el *spam* del correo, es preferible minimizar los falsos positivos para no eliminar correos importantes.

Adicionalmente, estos resultados obtenidos de la clasificación binaria son la base de otros conceptos esenciales para caracterizar el modelo y que posteriormente se usan en las métricas de evaluación del modelo. Algunos conceptos importantes son los siguientes: la **sensibilidad** del modelo indica la capacidad del modelo de acertar marcando los casos positivos en proporción con todos los casos que han señalado como positivos, es decir, se calcula SEN = TP/(TP+FN). La **especificidad** mide la capacidad del modelo de señalar casos negativos aquellos que no lo son en proporción con los casos negativos marcados correctamente, se calcula ESP = TN/(FP+TN). La **precisión** mide la capacidad del modelo de marcar correctamente los positivos sobre el total de positivos, PRECISION = TP/(TP + FP). Los casos positivos también se llaman relevantes, por ello, la precisión se puede definir también como la fracción de las instancias recuperadas que son relevantes y la sensibilidad la fracción de instancias relevantes que han sido correctamente recuperadas [33]. La **exactitud** mide la capacidad del modelo de acercarse a los valores reales, ACC = (TP + TN)/(P + N). No obstante, como hemos mencionado anteriormente, no nos podemos limitar a minimizar la exactitud del modelo ya que podremos tener un exceso de falsos positivos o falsos negativos y que nos interese conseguir el mejor balance posible, es por esto que surgen las métricas de evaluación del modelo.

Entre las métricas de evaluación hay que destacar dos: el ***F-score*** nos ayuda a mejorar la exactitud del modelo variando la precisión y la sensibilidad del modelo.

$$F_\beta = \left(1+\beta^2\right) \cdot \frac{\text{PRECISION} \cdot \text{SENSIBILIDAD}}{(\beta^2 \cdot \text{PRECISION}) + \text{SENSIBILIDAD}} \tag{3.4.41}$$

También se puede escribir en función de los valores de la matriz de confusión:

$$F_\beta = \frac{(1+\beta^2) \cdot \text{TP}}{(1+\beta^2) \cdot \text{TP} + \beta^2 \cdot \text{FN} + \text{FP}}. \tag{3.4.42}$$

Generalmente se suelen usar los valores $\beta = 1$, $\beta = 2$ y $\beta = 0{,}5$. El F_1 (3.4.43) ($\beta = 1$) se emplea en casos donde los falsos positivos reciben igual peso que los falsos negativos. Se calcula mediante una media armónica de la precisión y la sensibilidad. Sin embargo con la métrica F_2 demos una mayor importancia a la sensibilidad del modelo consiguiendo reducir la cantidad de falsos negativos. Mientras que la métrica $F_{0,5}$ de otorga un mayor peso a la precisión del modelo logrando disminuir los falsos positivos.

$$F_1 = \frac{2 \cdot \text{PRECISION} \cdot \text{SENSIBILIDAD}}{\text{PRECISION} + \text{SENSIBILIDAD}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}. \tag{3.4.43}$$

Por último, nótese que la métrica *F-score* no incluye los verdaderos negativos (TN) por ello en la práctica se suelen usar otras métricas como la κ de Cohen (3.4.44). Esta métrica incluye la posibilidad de acierto en las predicciones por azar.

$$\kappa = \frac{2 \times (\text{TP} \cdot \text{TN} - \text{FN} \cdot \text{FP})}{(\text{TP} + \text{FP}) \cdot (\text{FP} + \text{TN}) + (\text{TP} + \text{FN}) \cdot (\text{FN} + \text{TN})} \tag{3.4.44}$$

El método `train` también devuelve la verosimilitud del modelo entrenado y en caso de que se le pasen los valores de los estados, se devolverá la métrica F-score.

Por otro lado, también se suele usar la curva de características operativas del receptor, *Receiver operating characteristic* (**ROC**). Esta curva se crea al comparar la tasa de verdaderos positivos (sensibilidad) frente a la tasa de falsos positivos. El objetivo al trabajar con esta métrica es maximizar el área bajo la curva (AUC) que toma valores entre 0 y 1, siendo 1 el valor óptimo que indica un modelo perfecto que clasifica todas las instancias correctamente. En otras palabras, AUC mide la capacidad del modelo de clasificación para distinguir entre instancias positivas y negativas, pero a diferencia de *F-score*, la métrica AUC no minimiza directamente los falsos positivos o falsos negativos.

Otra forma de medir la calidad de la predicción es mediante el uso de funciones de pérdida. Estas funciones de pérdida nos aportan una medida cuantitativa a la hoja de calcular el error que hemos hecho en nuestra predicción. A continuación mencionaremos la más significativa en las clasificaciones binarias[4].

- *Cross-Entropy* binario (BCE): Mide la discrepancia entre las predicciones binarias obtenidas de la salida del modelo y los valores reales. En otras palabras, mide la diferencia entre la probabilidad predicha de que una muestra pertenezca a la clase positiva y la probabilidad real de que pertenezca a esa clase.

$$\mathrm{BCE}(y, p(s)) = -y \log(p(s)) - (1 - y) \log(1 - p(s))$$

  Donde y es el valor real, en nuestro caso `NBER`, y $p(s)$ es la probabilidad predicha por el modelo de que el resultado sea la clase positiva. No obstante, en este caso tratado tenemos un conjunto de datos, por lo que la función de pérdida global será

$$\mathrm{BCE} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(p(s_i)) + (1 - y_i) \log(1 - p(s_i))] \tag{3.4.45}$$

  La función de pérdida es igual a cero si el modelo predice la etiqueta correcta con una probabilidad del 100 % y a medida que la probabilidad predicha se aleja del valor real, la pérdida aumenta.

Hay que mencionar que generalmente usaremos las funciones de pérdida para calcular la tasa de error que cometemos en las probabilidades de transición o en los parámetros de la distribución de emisión, que afectan de manera implícita a las predicciones del modelo, en lugar de medir directamente la precisión de las predicciones. Luego usaremos los valores de las funciones de pérdida para buscar los valores óptimos de los parámetros. Y por otro lado, las métricas las usaremos para conocer la calidad del modelo.

[4]El universo de las funciones de pérdida es gigantesco y para cada problema que queremos modelar existen múltiples alternativas a escoger. Por ejemplo, para problemas de regresión existen el error cuadrático o el error absoluto. No obstante, la magia de crear un modelo que devuelva un mejor resultado reside en elegir una función de pérdida que aumente las fortalezas del modelo y disminuya las debilidades

3.4.5. Consideraciones adicionales

A raíz de estos conocimientos básicos existen una miríada de variaciones que con la práctica ayudan a perfeccionar las estructuras del modelo oculto de Markov elegido para resolver un problema, algunos destacables son los HMM autorregresivos, los HMM condicionalmente aleatorios, HMM con transiciones nulas o *Tied-HMM*.

Además existen algunas cuestiones que no hemos desarrollado a lo largo de este apartado, por ejemplo, qué hacer cuando tenemos múltiples observaciones, cuando tenemos datos erróneos o no tenemos suficientes datos para entrenar el modelo. Todas estas son cuestiones que quedan fuera del ámbito de este trabajo.

Selección de modelos.

Antes de comenzar con el ejemplo práctico, es necesario comentar que en los modelos de mixturas y en particular en los HMM, el aumento del número de estados conlleva un ajuste del modelo (hablamos de un ajuste en la verosimilitud). Pero a su vez pagamos un precio por ello, aumentando del número de parámetros al cuadrado. Por ello surgen algunos criterios para seleccionar la cantidad de estados que tiene el modelo.

En HMM existen dos técnicas de selección del número de estados de un modelo oculto de Markov según Langrock et al.[36]. Por un lado Akaike information criterion (AIC) sigue un enfoque frecuentista a la hora de resolver este problema. La lógica es que se asume un vector de observaciones $O_1, O_2, ..., O_N$ que fueron generados por un modelo "bueno" f desconocido y ahora con esas observaciones se ajustan dos familias de modelos $\{g_1 \in G_1\}$ y $\{g_2 \in G_2\}$. Entonces el objetivo es conocer cuál de las dos familias tiene una menor discrepancia con respecto a f, sin embargo al ser f desconocido debemos calcular las esperanzas de estas discrepancias $\widehat{\mathrm{E}}_f\left(\Delta\left(f, \widehat{g}_1\right)\right)$ y $\widehat{\mathrm{E}}_f\left(\Delta\left(f, \widehat{g}_2\right)\right)$. Los precursores, Linhart y Zucchini [15], usaron la discrepancia la divergencia K-L para acuñar el método de selección de modelo AIC:

$$\mathrm{AIC} = -2\log L + 2p$$

El método intenta establecer una balanza entre la calidad del ajuste del modelo, usando la log-verosimilitud, L, y la cantidad de parámetros p que tiene el modelo. El modelo HMM con el valor AIC más bajo se considera el mejor modelo, ya que proporciona el mejor equilibrio entre el ajuste de los datos y la complejidad del modelo.

Por otro lado, el enfoque bayesiano conocido como criterio de información bayesiano (BIC), consiste en establecer una probabilidades *a priori* $P(f \in G_1)$ y $P(f \in G_2)$ para calcular las probabilidades a posteriori $P(f \in G_1|\boldsymbol{x}^{(T)})$ y $P(f \in G_2|\boldsymbol{x}^{(T)})$.

$$\mathrm{BIC} = -2\log L + p \cdot \log T$$

donde T es el número de observaciones, en este caso si $T > \exp^2$ existe una mayor penalización a la hora de aumentar el número de parámetros p.

3.5. Límites de la aplicación y cómo ampliarla

Aunque la solución propuesta está enfocada a la generalidad, es difícil cubrir todos los casos de uso en los que se podría aplicar un modelo oculto de Markov. Por ello, en la siguiente sección vamos a discutir algunas de las limitaciones de la solución y cómo se puede ampliar esta para cubrirlas.

En primer lugar, como ya se ha mencionado, únicamente se ha construido un modelo oculto de Markov en el que las distribuciones de emisión son distribuciones normales. Entonces, si se quiere crear un modelo oculto de Markov cuya distribución de emisión sea distinta a una distribución normal, por ejemplo Poisson, habrá que crear una clase llamada `PoissonHMM` similar a las clases `CategoricalHMM` y `NormalHMM`. Además habrá que sobreescribir los métodos específicos que varían para cada distribución (`label_train`, `initialize_parameters`, `conditional_dist`, `log_train_baum_welch`, `train_baum_welch` y `forecast_dist`) además de cambiar las distribuciones para cada estado (el parámetro `distributions`), para esto último nos podremos apoyar en el módulo `Distributions` de la biblioteca `TensorflowProbability` que implementa las características básicas de muchas distribuciones de probabilidad.

Por otro lado, en la implementación considerada tendremos una distribución por cada estado de la cadena oculta de Markov y esta distribución no cambiará con el tiempo. Esta solución, conocida como *Switching Markov Model* es interesante para modelar situaciones en la que existe una autocorrelación en los datos. En el anexo (sección 6.2) se desarrolla este concepto en detalle y su implementación pasa nuevamente por crear una clase `ArHMM` que herede de `HMM`. En este sentido, existen una infinidad de modificaciones que se le pueden realizar al modelo oculto de Markov, como tener varias cadenas ocultas de Markov o tener cadenas de Markov de orden superior (cadenas que en lugar de depender únicamente del estado anterior, dependen de n estados anteriores).

También, existe otra limitación en relación con la cadena oculta de Markov del HMM. Esta cadena tiene que ser discreta, sin embargo a veces puede ser útil modelar procesos estocásticos de tiempo continuo. Por ello, se podría transformar la implementación del modelo oculto de Markov a un modelo espacio-estado más general. En este caso, la solución realizada tendría que cambiar por completo, ya que los cálculos de las probabilidades cambiarían completamente.

Finalmente, algo más concreto de esta implementación es la división del conjunto de datos. Como ya hemos comentado en la sección 3.4.4, normalmente se evalúa el modelo usando *cross-validation*, pero en la implementación actual se ha dividido el conjunto de datos respetando el orden. El motivo es que al no haber identificado los modelos cuyas observaciones tienen una correlación, al variar el orden la perderíamos y el modelo produciría valores incorrectos.

Capítulo 4

Caso de uso: Modelo de predicción de recesiones económicas.

Como hemos avanzado en la introducción del trabajo, vamos a aplicar la teoría vista en el apartado anterior sobre modelos ocultos de Markov, para modelar la evolución de la economía estadounidense e intentar prever una recesión. El objetivo de la siguiente sección es mostrar una aplicación práctica de los modelos ocultos de Markov haciendo uso de la biblioteca creada anteriormente.

A modo de resumen, en este capítulo compararemos varios modos de entrenamiento y veremos sus resultados. Por un lado los entrenamientos supervisado y no supervisado implementados en la biblioteca (sección 3.4.3 y sección 3.4.3) y por otro, entrenamientos realizados utilizando la biblioteca `TensorflowProbability` (TFP):

Supervisado v.s. no supervisado	**Desarrollador del algoritmo**	**Método de estimación**	**Escalado**	**Tabla**	**Identificador**
Usando NBER (SUPERVISADO)	TFP	Maximiza la log-verosimilitud	NO HAY	Tabla 6	TFP log-verosimilitud
		Minimiza BCE	NO HAY	Tabla 6	BCE
	Propio	Estima la log-verosmilitud	NO HAY	Tabla 6	Estadísticos Suficientes
Sin NBER (NO SUPERVISADO)	TFP	NO HAY	NO HAY	NO HAY	NO HAY
	Propio	Baum-Welch con datos escalados	Factor de escalado	Tabla 8	Baum-Welch factor escalado
			Escalado logarítmico	Tabla 8	Baum-Welch escalado logarítmico

Tabla 4: Tabla resumen con los algoritmos de entrenamiento implementados, indicando si el entrenamiento es supervisado o no supervisado, el desarrollador del algoritmo (`TensorflowProbability` o código propio), la tabla en la que aparecen las métricas de evaluación F1-*score* y F2-*score* y el identificador usado en el trabajo.

4.1. Datos

A continuación comentaremos el tratamiento necesario de los datos. Para este trabajo no será necesario extraer datos de una fuente externa, sino que se considerarán ya integrados en el problema. Más concretamente, los datos estarán ya guardados en una tabla de *Excel*. Por lo tanto, únicamente realizaremos la etapa de transformación de los datos.

4.1.1. Descripción y análisis del conjunto de datos.

El conjunto de datos original, contiene el producto interior bruto (PIB) de Estados Unidos en billones de dólares americanos, la fecha en la que se registró ese producto interior bruto (Date), y una variable de control (`NBER`). En concreto, tendremos cuatro registros de datos al año, uno por cada trimestre desde 1947, enumerados de la siguiente forma:

	Date	GDP	NBER
0	1947.1	2034.450	0
1	1947.2	2029.024	0
2	1947.3	2024.834	0
3	1947.4	2056.508	0
4	1948.1	2087.442	0

Figura 13: Primeros cinco registros temporales de los datos originales de producto interior bruto.

Para poder considerar la serie del producto interior bruto a lo largo del tiempo, es necesario que la distancia entre los índices sea la misma. Es decir, que para cuales quiera índices temporales $i = 1, 2, ..., T$, $\text{Date}(t_{i+1}) - \text{Date}(t_i) = 1/4$ sea constante y no exista una mayor distancia entre el último trimestre y el primero del año siguiente. Por lo tanto, transformaremos los índices temporales al formato "año.fracción" donde "fracción" es la fracción del año correspondiente al trimestre. De esta manera, se logra una distancia temporal constante entre los índices.

	Date	GDP	NBER
0	1947.25	2034.450	0
1	1947.50	2029.024	0
2	1947.75	2024.834	0
3	1948.00	2056.508	0
4	1948.25	2087.442	0

Figura 14: Primeros cinco registros temporales de los datos con los tiempos modificados.

Ahora podemos dibujar la gráfica que sigue la evolución del producto interior bruto respecto del tiempo (Figura 17). Como vemos hay una tendencia alcista, con algunos

momentos en los que la economía americana se ha resentido un poco. La principal razón por la que el producto interior bruto de los países crece es porque vivimos en una economía capitalista que promueve un rápido avance tecnológico[27], pero existen otras múltiples razones que influyen en la variación del PIB como puede ser el crecimiento de la población o políticas gubernamentales[6][34].

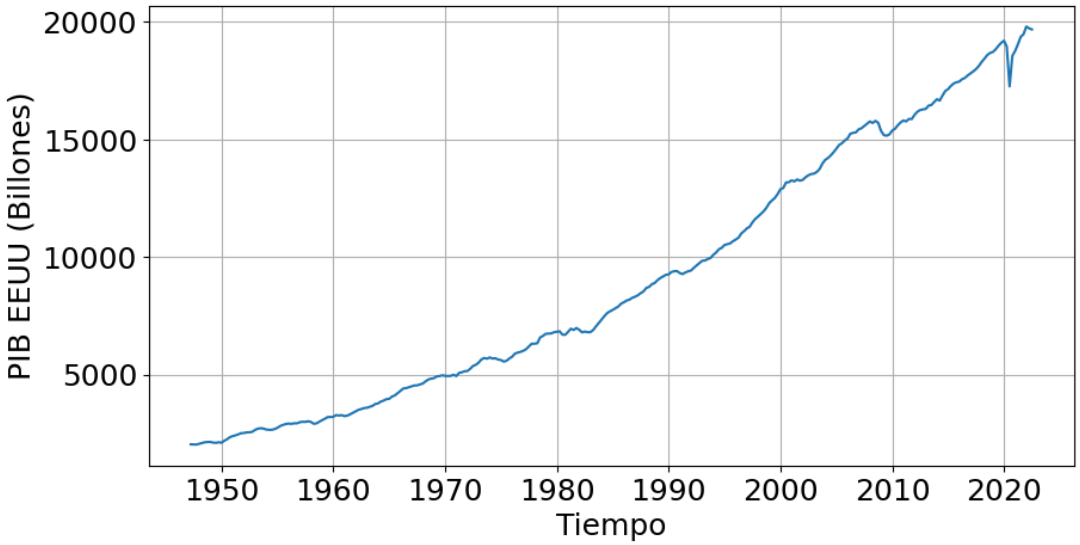


Figura 15: Crecimiento del producto interior bruto estadounidense con respecto del tiempo.

No obstante, en nuestro modelo queremos capturar el estado del mercado, si nos encontramos en un periodo de crecimiento o de recesión, por ello si consideramos la variación del PIB en cada trimestre podremos ver una posible segregación en dos estados. Al fin y al cabo, si consideramos la variación veremos la volatilidad y la tendencia de los últimos trimestres y esto nos va a ayudar directamente a identificar mejor si estamos en una recesión. Como hemos visto en la introducción, la economía tiene un comportamiento cíclico en el que existe un periodo de expansión donde la variación será positiva, luego tendremos un máximo de crecimiento antes del periodo de contracción, en el cual veremos una variación cercana a cero y finalmente en las recesiones tendremos una variación negativa.

Como el crecimiento de la economía es exponencial (y crece, al menos, al ritmo de la inflación), es decir que si aumenta el PIB un 2 %, lo hace respecto al último valor, y así sucesivamente. Esto se conoce como tasa de crecimiento continuo , y nos ayuda a medir el cambio de una variable, en este caso el PIB, en relación con el tiempo.

$$\text{tasa} = (\ln(O_{t+1}) - \ln(O_t)) \cdot 100 \tag{4.1.1}$$

Guardaremos este valor en la variable Change. Como para el valor t_0 no existe una variación, simplemente lo inicializaremos a 0.

	Date	GDP	Change	NBER
0	1947.25	2034.450	0.000000	0
1	1947.50	2029.024	-0.267062	0
2	1947.75	2024.834	-0.206717	0
3	1948.00	2056.508	1.552168	0
4	1948.25	2087.442	1.492999	0

Figura 16: Primeros cinco registros de los datos que se usarán para predecir los estados de crecimiento o recesión.

Gráficamente, vemos la variación de la siguiente forma (Figura 17):

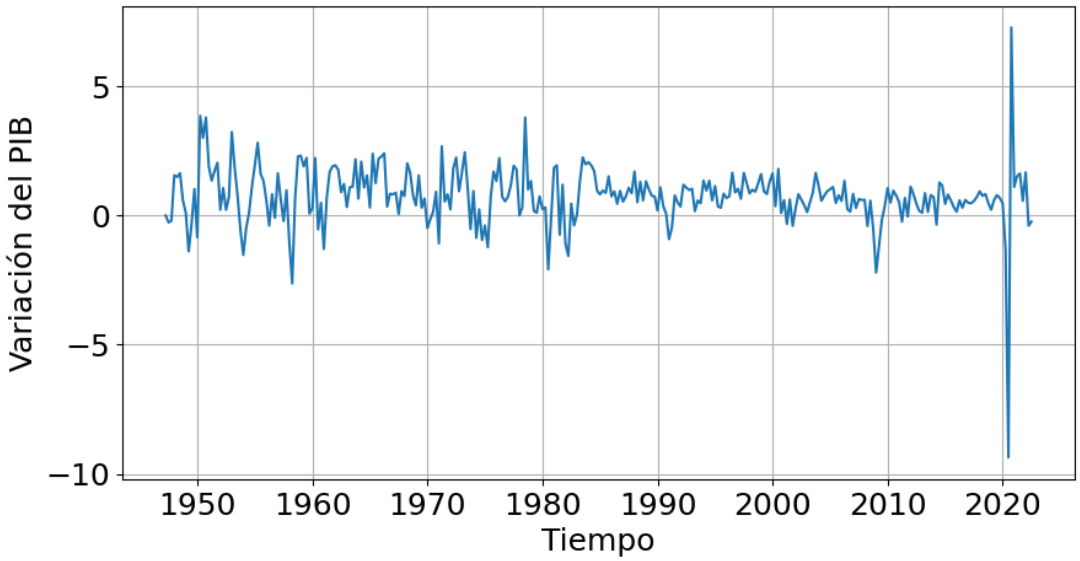


Figura 17: Variación logarítmica del producto interior bruto .

Finalmente, hay que mencionar que para este problema disponemos de los tiempos en los que ha habido una recesión. Guardamos en la variable `NBER` el valor 1 si existe recesión y 0 en caso contrario. Gracias a esto, podremos realizar también un entrenamiento supervisado.

Es evidente que este etiquetado lo podemos hacer únicamente *a posteriori*, ya que no existe una forma de predecir con exactitud si en cinco años habrá una recesión. Gracias a esta variable `NBER` podemos ver gráficamente (Figura 18) en qué momentos se produjeron estas recesiones:

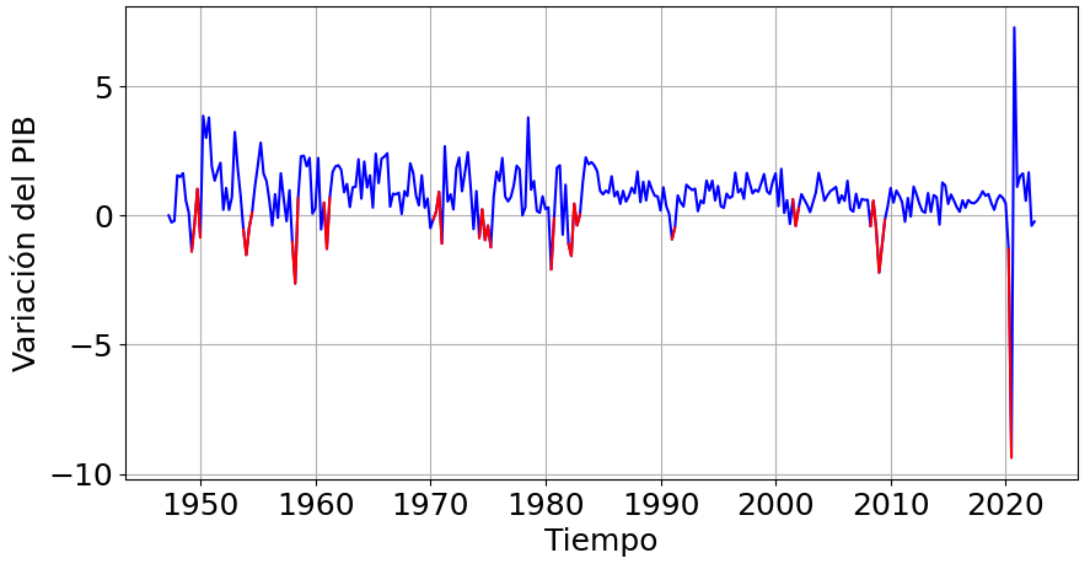


Figura 18: Variación logarítmica del producto interior bruto con los intervalos de recesión marcados en rojo.

4.2. Construcción del HMM

En esta sección construiremos gradualmente el modelo oculto de Markov para predecir si existe una recesión en la economía o no. Para ello, utilizaremos los fundamentos teóricos que hemos establecido previamente sobre los modelos de mixturas y los modelos ocultos de Markov. Empezaremos prestando atención a la variable cambio, definiremos los estados ocultos y observables, y luego trataremos de usar los métodos definidos previamente para estimar los parámetros.

4.2.1. Identificación de la cantidad de estados ocultos.

En primer lugar, partiremos de la suposición de que la variabilidad se distribuye de forma normal con media μ y desviación típica σ, esta asunción es bastante acertada ya que en general los modelos financieros y autorregresivos sufren de curtsosis, es decir, existe un punto de la función de densidad con excesivo achatamiento y concentración de valores. Otras distribuciones como la distribución *t-student* podrían ser buenas opciones también. En la siguiente gráfica podemos ver un histograma que incluye ambos estados (Figura 19).

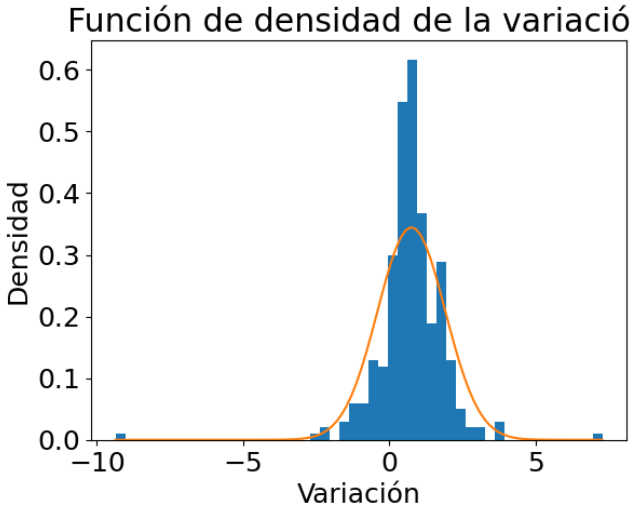


Figura 19: Histograma con la variación del PIB en todos los instantes del estudio. Además se ajusta una distribución normal con la media y varianza muestral.

Puesto que el objetivo es identificar si existe una recesión o no, nos centraremos en construir un modelo oculto de Markov con dos estados ocultos de tal forma que cada estado oculto emite un valor que distribuye normal(μ_i, σ_i), según $i = 0$ (no recesión), $i = 1$ (recesión). Esto implica que tendremos un modelo de mixtura con dos distribuciones normales que traten de ajustar la doble moda que existe en el histograma anterior (Figura 19). Una primera aproximación para ajustar las dos distribuciones normales, aprovechando que tenemos datos supervisados, podría ser estimar los parámetros de una distribución normal (usando media y varianza muestral) con los valores en los que $\texttt{NBER}(t_i) = 0$, y por otro ajustar otra distribución normal cuyos valores sea $\texttt{NBER}(t_i) = 1$. Obteniendo el siguiente resultado (Figura 20):

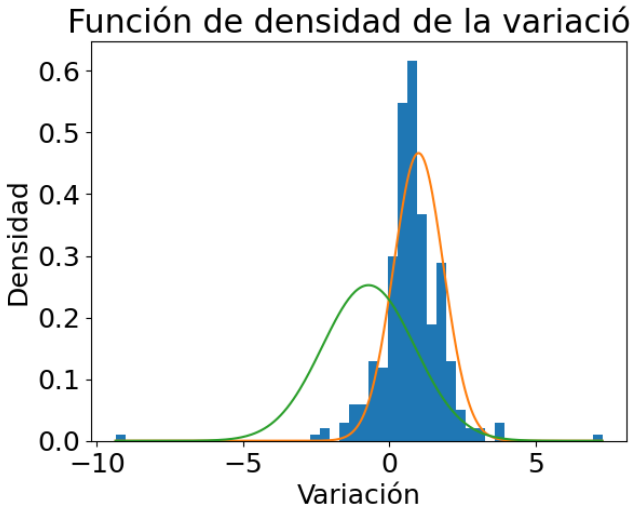


Figura 20: Histograma con la variación del PIB en todos los instantes del estudio. En este caso se ajustan dos distribuciones.

Como podemos observar, este enfoque es bastante equivocado y no concuerda realmente con las variaciones observadas en el histograma. Esto puede ser por múltiples

motivos: en primer lugar, como es normal, los estados marcados como "estado de recesión" ($\texttt{NBER}(t_i) = 1$) son estados en los que existe una variación negativa o pequeña con respecto al tiempo anterior. Esto se puede observar en la Figura 18, en el que la mayoría de instantes en el tiempo marcados en rojo se sitúan en mínimos locales y en valores negativos. Sin embargo, la realidad es que los estados de recesión ocurren con cierto retraso, es decir, existe un paso previo a una recesión en el que la economía se llega a ralentizar pero no entra en recesión. Estos estados, aún marcados como ($\texttt{NBER}(t_i) = 0$) podrían perfectamente pertenecer a un tercer estado en el que no existe un crecimiento. En cualquier caso, nuestra aproximación no incluye en ningún caso estos estados dudosos como "estados de recesión". Otro motivo podría ser que únicamente tenemos una secuencia de cambios, es decir, no tenemos un conjunto de datos infinitos de toda la evolución del PIB durante la existencia del ser humano y de tiempos futuros. Por ello, la estimación será a lo sumo como los datos del estudio.

Por último, existe una forma de medir la bondad del ajuste realizando un diagnóstico del modelo o GoF Test (*Goodness of Fit Test*). Medir la bondad del ajuste implica ver las diferencias existentes entre los valores observados (Change(t_i)) y la normal(μ, σ) que es como suponemos que se distribuyen las variables. En concreto, podríamos realizar un Kolmogorov-Smirnov Test (KS Test) para validar que en efecto, las variaciones siguen una distribución normal. Este test mide la distancia vertical entre los datos, es decir para instante t_i con $i = 1, 2, ..., T$, comparamos el valor de Change(t_i) con el valor de la distribución normal(μ, σ) de la siguiente forma:

$$D = \sup_x |F(x) - Change_{t_i}(x)| \tag{4.2.1}$$

donde $F(x)$ es la normal(μ, σ). Visualmente calculamos los siguientes valores para ambos casos, cuando hay recesión y cuando no la hay.

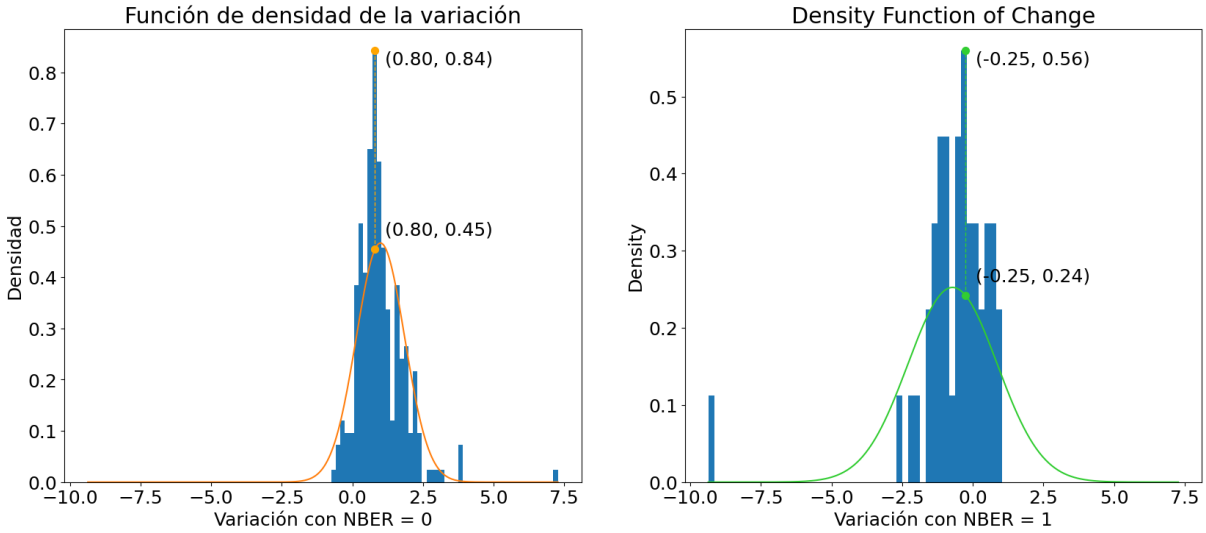


Figura 21: Distancias entre las funciones de densidad y el histograma para cada uno de los casos, cuando existe recesión y cuando no la hay.

A raíz de lo anterior podemos hacer un test de hipótesis nula rechazando la hipótesis principal si el valor D de (4.2.1) supera el valor crítico.

```
KSTest (NBER = 0): KstestResult(statistic=0.4790713135083996,
   pvalue=1.6290603683426422e-29, statistic_location
   =0.09388324015880606, statistic_sign=-1)
KSTest (NBER = 1): KstestResult(statistic=0.6976744186046512,
   pvalue=1.7945884348739528e-18, statistic_location
   =-0.11891779649300105, statistic_sign=1)
```

Listado de código 4.1: Resultado de realizar el KS test.

No obstante, al ser datos en el tiempo, existe una autocorrelación en los datos, y por lo tanto hacer un GoF no sería del todo correcto ya que este asume que los datos son independientes e idénticamente distribuidos. Asimismo, como los datos están autocorrelacionados los resultados obtenidos 4.1 están sesgados y por lo tanto hemos obtenido un p-valor incorrecto.

En este caso, tendría más sentido evaluar el ajuste del modelo utilizando los residuos para verificar su capacidad de representar los datos. Esta evaluación se llevará a cabo después de entrenar el modelo oculto de Markov para determinar cuál de todos los modelos se ajusta mejor a los datos.

4.2.2. Estimación de los parámetros del HMM.

Ahora vamos a proceder a construir el modelo oculto de Markov, definiendo los dos estados ocultos existentes, el vector de probabilidades iniciales, la matriz de transición de estados y la matriz de emisión de probabilidades. En las siguientes secciones trataremos los dos algoritmos implementados en la solución: el algoritmo supervisado y el algoritmo de Baum-Welch.

Enfoque Supervisado

El aprendizaje de un modelo oculto de Markov no suele ser supervisado ya que el valor del estado oculto no suele observarse en la vida real, la biblioteca implementa un método de entrenamiento supervisado. En este caso, haremos uso de los datos de la variable `NBER`. Posteriormente, compararemos este modelo con otros entrenados usando el algoritmo de Baum-Welch (Sección 3.4.3).

El algoritmo estima las probabilidades iniciales, la matriz de transición y los parámetros de las distribuciones de emisión haciendo uso de las ecuaciones mencionadas anteriormente (3.4.36, 3.4.37, 3.4.39 y 3.4.40). Ejecutando el entrenamiento utilizando el 70 % de los datos para entrenar y evaluando el 30 % restante con las métricas F_1-*score* y F_2-*score*, obtenemos:

```
F1-score: 0.8421
F2-score: 0.7692
```

Listado de código 4.2: Resultado de evaluar el modelo cuyos parámetros se han estimado con estadísticos suficientes usando la métrica F_1-*score* y F_2-*score*.

Podemos hacer una predicción del conjunto de datos completo, y aunque el modelo ya conozca el 70 % de los datos, gráficamente vemos (Figura 22) cómo comete algunos errores:

Figura 22: Gráfica con la evolución del PIB con respecto del tiempo. Los colores indican la variación de los estados latentes tanto predichos por el modelo (color verde) como etiquetados por la variable `NBER` (color rojo). Esta cadena es la más probable con una probabilidad de 2,0173 $e-218$.

A continuación podemos ver (Tabla 5) la matriz de confusión con los aciertos y los errores en la predicción anterior (Figura 22):

		Predicciones	
	Población total = P + N	Positivos (PP)	Negativos (PN)
Estados	Positivos (P)	256	3
Actuales	Negativos (N)	9	34

Tabla 5: Matriz de confusión con los resultados de la clasificación binaria. De los 302 instantes de tiempo, se han predecido correctamente 34 recesiones, 256 casos en los que no había recesión, y hemos cometido 9 falsos positivos y 3 falsos negativos.

Antes de proceder al siguiente enfoque, hay que mencionar que `TensorFlowProbability` nos ofrece la posibilidad de entrenar los parámetros de las distribuciones de emisión usando un optimizador de la clase `tensorflow.optimizers`, en concreto usaremos el optimizador ADAM (*Adaptive Moment Estimation*)[29].

Como en este enfoque supervisado estamos haciendo uso de las etiquetas, podemos llevar a cabo dos optimizaciones: en la primera optimización estableceremos como función objetivo la log-verosimilitud y realizaremos 100 iteraciones del algoritmo de optimización

tratando de minimizarla e intentando unos parámetros que mejoren las predicciones. Por otro lado, podemos hacer una optimización minimizando la función de pérdida *cross-entropy* binaria con el optimizador ADAM.

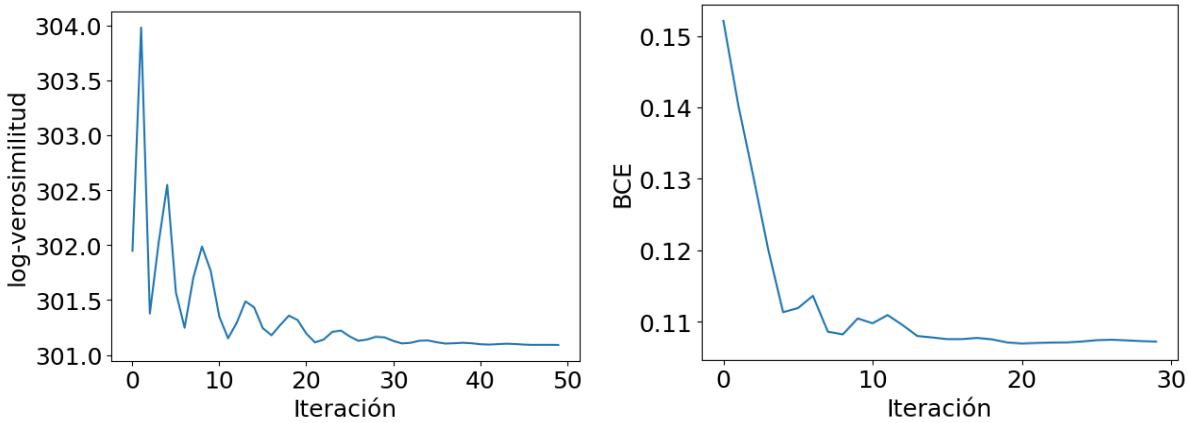


Figura 23: El gráfico de la izquierda representa gráficamente el proceso de optimización de la verosimilitud, mientras que el gráfico de la derecha muestra el proceso de optimización de la minimización de la métrica BCE.

A continuación se muestra una comparativa de los tres métodos usados (Tabla 6):

	Estadísticos suficientes	TFP log-verosimilitud	BCE
Media	[1.1152, -0.52]	[1.1593, -0.2519]	[0.973 , -0.9249]
Desv. Est.	[0.8192, 0.8594]	[0.8164 , 0.9048]	[0.5996, 0.9398]

Tabla 6: Comparación entre los tres modelos creados, el primero estima los parámetros de las distribuciones de emisión (μ y σ) usando los estadísticos suficientes (3.4.39, 3.4.40), el segundo calcula la optimización de la log-verosimilitud para buscar los parámetros óptimos y el tercero calcula la optimización de la métrica *cross-entropy* binaria para obtener los parámetros óptimos.

Como vemos en la Tabla 7, los resultados obtenidos tras la optimización de los parámetros usando la log-verosimilitud son casi idénticos al modelo que hace uso de estadísticos suficientes para estimar los parámetros. Mientras que la opción que trata de optimizar la métrica *cross-entropy* binaria no consigue tan buenos resultados. Esto se puede deber a que al usar directamente las etiquetas para conseguir unos parámetros que ajusten mejor el modelo a los datos, el modelo termina sobre-ajustando al 70 % del conjunto de datos y por tanto, cuando va a evaluar el 30 % restante no reproduce los datos con la misma precisión. Para resolver este sobre ajustes se pueden usar técnicas como la validación cruzada progresiva y otras que veremos en la sección de trabajos futuros.

	Estadísticos suficientes	TFP log-verosimilitud	BCE
F1-score	0.8421	0.8421	0.7273
F2-score	0.7692	0.7692	0.7273

Tabla 7: La tabla muestra la diferencia de las métricas F1-*score* y F2-*score* entre los tres modelos creados. Los tres modelos usan un apredizaje supervisado.

Enfoque Baum-Welch

El siguiente enfoque que adoptaremos para estimar los parámetros del modelo es el algoritmo de maximización de las expectativas que vimos en la sección (3.4.3). A diferencia del apartado anterior, en este caso no haremos uso de las etiquetas `NBER` salvo para calcular la calidad de las predicciones. Como hemos comentado, este es un enfoque que se aproxima más a la realidad pues normalmente no disponemos de los datos etiquetados.

Ahora llamamos al método `train` indicando que queremos entrenar el modelo usando el algoritmo de Baum-Welch.

	Estadísticos Suficientes	Baum-Welch escalado logarítmico	Baum-Welch factor escalado
π	[0.8483, 0.1517]	[0, 1]	[1, 0]
A	[[0.9497, 0.0503], [0.2812, 0.7188]]	[[0.917 0.0962], [0.2669 0.7675]]	[[0.9214 0.0786], [0.2482 0.7518]]
Media	[1.1152, -0.52]	[1.2276 -0.0766]	[1.2076, -0.1378]
Desv. Est.	[0.8192, 0.8594]	[0.794 0.9105]	[0.3039, 0.9039]

Tabla 8: La tabla muestra la diferencia entre los parámetros resultantes del algoritmo supervisado, que estima los parámetros λ mediante estadísticos suficientes, expuesto en la tabla anterior (4.2.2) y las dos implementaciones del algoritmo de Baum-Welch. π es el vector de probabilidades iniciales, A la matriz de transición y los valores "media" y "Desv. Est." corresponden a los parámetros μ y σ de la distribución de emisión.

A continuación vamos a comparar el resultado anterior, obtenido mediante los estadísticos suficientes (Tabla 9), con el algoritmo Baum-Welch. Basta con compararlo con el resultado de nuestra implementación (estadísticos suficientes) ya que es tan bueno como la implementación de `TensorflowProbability`. Vemos que los resultados no mejora las puntuación F_1-*score* pero sí consiguen alcanzar aproximadamente el mismo valor en la puntuación F_2-*score*.

	Estadísticos Suficientes	Baum-Welch escalado logarítmico	Baum-Welch factor escalado
F1-score	0.8421	0.7097	0.7273
F2-score	0.7692	0.8594	0.7273

Tabla 9: La tabla muestra la diferencia de las métricas F1-*score* y F2-*score* entre los tres modelos creados.

También podemos añadir las matrices de confusión para el algoritmo Baum-Welch con escalado logarítmico (Tabla 10a) y Baum-Welch con factor de escalado (Tabla 10b).

		Predicciones	
		(PP)	(PN)
Estados	(P)	41	2
Actuales	(N)	20	239

(a) Baum-Welch escalado logarítmico

		Predicciones	
		(PP)	(PN)
Estados	(P)	38	5
Actuales	(N)	14	245

(b) Baum-Welch factor de escalado

Tabla 10: En la tabla (a) podemos ver la matriz de confusión con los resultados de la clasificación usando Baum-Welch escalado logarítmico. De los 302 instantes de tiempo, se han predecido correctamente 41 recesiones, 239 casos en los que no había recesión, y hemos cometido 20 falsos positivos y 2 falsos negativos. Mientras que en la clasificación usando Baum-Welch factor de escalado, se han predecido correctamente 38 recesiones, 245 casos en los que no había recesión, y hemos cometido 14 falsos positivos y 5 falsos negativos.

No obstante, como ya se comentó en el apartado de teoría, la optimización de la función de verosimilitud puede resultar en un máximo local, en lugar de encontrar el valor óptimo global. Para solucionar esto, se ha implementado un *grid-search*, es decir creamos varios *arrays* que contienen las condiciones iniciales y para cada combinación de condiciones iniciales llamamos a la función de optimizar. De esta forma esperamos obtener otros máximos y con fortuna encontraremos el óptimo general. Hay que mencionar que este procedimiento es costoso en tiempo y no existe una garantía de mejorar la solución obtenida.

Después de realizar el grid-search, los parámetros iniciales que nos otorgan una mayor verosimilitud son:

$$\begin{aligned} \pi_0 &= [0{,}6, 0{,}4] \\ A &= [[0{,}6, 0{,}4], [0{,}6, 0{,}4]] \\ \mu &= [0{,}5, -0{,}5] \\ \sigma &= [1, 1] \end{aligned}$$

Sin embargo, los resultados obtenidos después de la optimización son muy similares a los conseguidos anteriormente 9. Esto es una indicación de que la optimización llevada a cabo es correcta.

Selección del modelo óptimo

La selección del mejor modelo parece una tarea sencilla, de todos los modelos obtenidos en el *grid-search* anterior, podemos escoger el que resulte en un mayor valor de verosimilitud (recordemos que en la realidad no disponemos de las etiquetas `NBER` que nos indiquen en qué estado estamos, así que no podemos calcular el *F-score* en cada iteración del *grid-search*). Sin embargo, todavía tenemos que validar el modelo, es decir necesitamos saber si la solución encontrada se ajusta correctamente e identificar posibles valores marginales (*outliers*).

Existen múltiples formas de validar la calidad del modelo, Buckby et. al [5] comentan algunas pero la más utilizada es la técnica de los pseudo-residuos. En los modelos tradicionales de regresión, medimos el ajuste usando los residuos, que representan la distancia entre el valor predicho por el modelo y el valor real. A continuación redefiniremos este concepto en el contexto de los HMM.

El pseudo-residuo ordinario usa la distribución condicional de una observación dadas todas las demás:

$$F(O_t|O_1, \ldots, O_{t1}, O_{t+1}, ..., O_T)$$

de esta forma compara los valores de estas distribuciones condicionales con el valor real. Asimismo, valoraremos la calidad del modelo en función de cómo se ajusten los residuos.

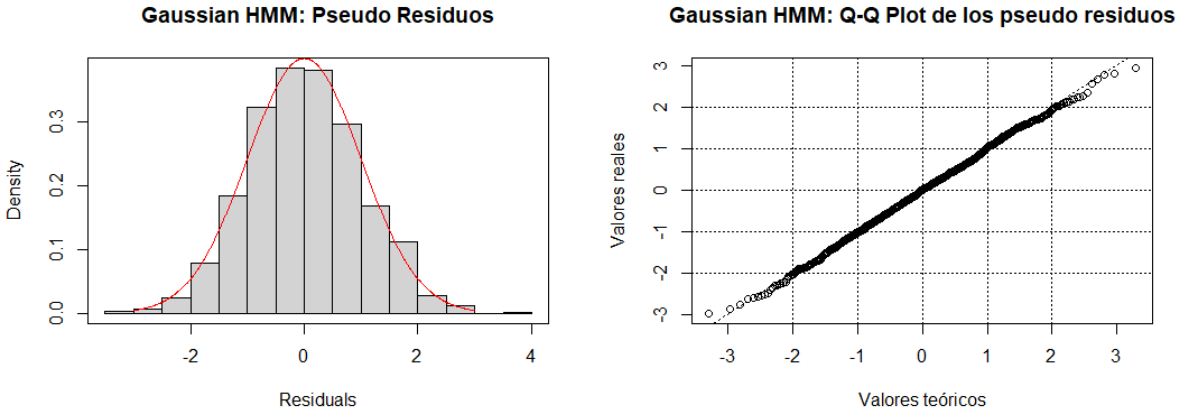


Figura 24: Histograma de los residuos con el ajuste de una normal con media 0 y varianza 1, y un gráfico de cuantiles (*Q-Q plot*) para ver las diferencias de probabilidad entre la normal y los valores de los residuos obtenidos.

Como vemos en la Figura 24, el modelo consigue ajustar perfectamente los residuos. Concluyendo que la distribución normal escogida al principio de la sección para modelar las distribuciones de emisión es correcta, y por otro lado los parámetros óptimos encontrados en el método de Baum-Welch son buenos.

4.2.3. Previsiones, decodificación y predicción de estados.

Ahora podemos hacer uso de nuevo del método `viterbi` de la biblioteca para calcular la cadena de estados más probable dados los parámetros óptimos obtenidos $\hat{\lambda}$.

Figura 25: Gráfica con la variación de los estados latentes tanto predichos por el modelo (color verde) como etiquetados por la variable `NBER` (color rojo). La probabilidad de observar la cadena dibujada en verde es de $3{,}3235\,e{-}221$.

También podemos predecir estados futuros usando el método `forecast_states`. El resultado de los siguientes tres años será que no habrá recesión con una probabilidad de $0,631$ para el primer año, $0{,}5786$ para el segundo y $0{,}5472$ para el tercero. Además podemos simular las distribuciones predictivas para los siguientes 20 instantes temporales (Figura 26):

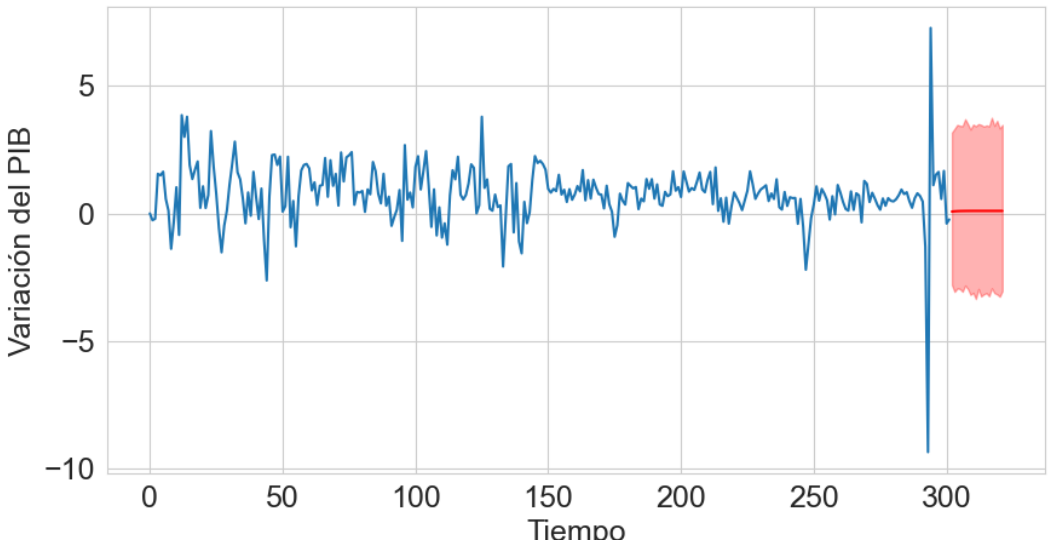


Figura 26: Gráfica con la variación del PIB estadounidense y las predicciones a futuro. En rojo se muestra el valor más probable y el intervalo de credibilidad del 95 %.

4.3. Ventajas de la biblioteca frente a TFP.

El objetivo de esta sección es ilustrar las diferencias principales de nuestro código frente a la biblioteca creada por Google, `TensorFlowProbability`. Si bien existen otras bibliotecas (mencionadas en la introducción), la comparativa con `TensorFlowProbability` es directa ya que durante el trabajo se han propuesto estimaciones de los parámetros del modelo oculto de Markov implementadas con esta biblioteca. Veamos a continuación (Figura 11) una comparativa de ambas bibliotecas:

	Tengo un modelo.	**Calcular las probabilidades *forward* y *backward*.**	**Calcular las probabilidades $\gamma_t(s_i)$**	**Calcular la probabilidad observar los datos dados los parámetros del modelo.**	**Calcular la cadena "oculta" (realmente la estoy observando) más probable hasta $t \leq T$.**	**Calcular la cadena oculta hasta T+h.**	**Estimar los parámetros del modelo oculto de Markov.**	**Estimar la cantidad de estados ocultos que consiguen el mejor ajuste.**
TFP con NBER	Sí	No	Sí	Sí	Sí	No	Sí	No
Propio con NBER	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí

	Tengo un modelo.	**Calcular las probabilidades *forward* y *backward*.**	**Calcular las probabilidades $\gamma_t(s_i)$**	**Calcular la probabilidad observar los datos dados los parámetros del modelo.**	**Calcular la cadena "oculta" (realmente la estoy observando) más probable hasta $t \leq T$.**	**Calcular la cadena oculta hasta T+h.**	**Estimar los parámetros del modelo oculto de Markov.**	**Estimar la cantidad de estados ocultos que consiguen el mejor ajuste.**
TFP sin NBER	No	No	No	No	No	No	No	No
Propio sin NBER	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí

Tabla 11: Comparativa de la funcionalidad de la biblioteca implementada en el trabajo frente a la bilioteca `TensorFlowProbability`.

En general, la biblioteca implementa los algoritmos más famosos a la hora de trabajar con modelos ocultos de Markov, y va más allá ofreciendo al usuario la posibilidad de obtener estimaciones de los estados ocultos pertenecientes a tiempos pasados y futuros. En ambos casos, la biblioteca entrega al usuario la probabilidad con la que ocurren estas estimaciones. Como se puede ver, la librería implementada en este trabajo es bastante exhaustiva, incluso más que la librería de `TensorFlowProbability`, permitiendo al usuario llevar a cabo más casos de uso.

Capítulo 5

Conclusiones

En el trabajo se ha intentado realizar una identificación de los estados de recesión en la economía, pero vemos que los resultados obtenidos no se asemejan a la perfección con las etiquetas reales. Esto puede ser por dos motivos: en primer lugar, los economistas han identificado estas recesiones con distintas métricas analizadas a posteriori, es decir, han etiquetado un periodo temporal como periodo de recesión gracias a métricas distintas al PIB y habiendo visto las consecuencias del periodo de recesión. En segundo lugar, al no tener un aprendizaje supervisado es más difícil identificar las características subyacentes de los datos que identifica la existencia de una recesión. Esto no quiere tampoco decir que los resultados obtenidos mediante el modelo optimizado usando los datos etiquetados sean los más precisos, pues esta precisión seguramente se podría mejorar usando otros modelos de clasificación supervisado, como redes neuronales u otros modelos autorregresivos. Sin embargo, el alcance de este trabajo limita el estudio de los estados de recesión a los modelos ocultos de Markov.

Por otro lado, hemos podido ver cómo en efecto dependiendo de los valores iniciales de los parámetros, el resultado del ajuste del modelo iba variando. También hemos podido observar que no existe una relación tan directa entre la verosimilitud de los datos y las *F-scores* ya que una mejora en el valor de la verosimilitud, no implica una mejora en las métricas *F-scores*.

5.1. Resultados

En cuánto a los resultados obtenidos, como vemos en el análisis a posteriori predictivo de la Figura 25 conseguimos identificar correctamente la mayoría de recesiones. No obstante, si nos fijamos en los valores de las métricas *F1-score* y *F2-score*, podemos ver que aún hay margen de mejora.

Actualmente vivimos en una situación de incertidumbre económica, por lo que podríamos utilizar el modelo para predecir si va a existir una recesión, como ya adelantamos en la sección 4.2.3. Las estimaciones puntuales del modelo apuntan que para el tercer trimestre de 2022 (recordemos que el último dato es del segundo trimestre de 2022) existe un 63.1 % de probabilidades de no estar en recesión y un 36.9 % de estarlo.

5.2. Fortalezas y aspectos de mejora

Como hemos mencionado, aunque las predicciones del modelo son "buenas" aún hay cosas que mejorar. En primer lugar, podríamos utilizar el enfoque bayesiano de los modelos ocultos de Markov para poder dar intervalos de confianza en lugar de estimaciones puntuales. Esto ayudaría a conocer en qué puntos tenemos una mayor certeza de que va a haber una recesión o no, esto último es útil para las predicciones.

Por otro lado, el universo de los modelos ocultos de Markov y los modelos espacio estado es inmenso. Aunque nos hemos centrado en los HMM normales, podríamos haber extrapolado el concepto a otras distribuciones como *t-student*. No obstante, aunque se ha desarrollado una solución ante un problema particular, la implementación llevada a cabo permite ser adaptada únicamente añadiendo una nueva clase en la que sólo habrá que cambiar las distribuciones de emisión.

Por último, otra fortaleza del enfoque usando HMM es que el aprendizaje llevado a cabo es no supervisado. Esto nos permite extrapolar la implementación de este modelo a situaciones de la vida real similares, es decir, situaciones en las que tengamos unos datos cuyas observaciones se distribuyen normal.

5.3. Trabajo futuro

Finalmente, existen varios caminos a seguir en caso de que se quiera continuar este trabajo. El primero y más claro sería programar los modelos ocultos de Markov autorregresivos y ver si consiguen un mejor ajuste del modelo. Por otro lado, se pueden implementar otros algoritmos de entrenamiento como el *segmental k-means* [22] o algoritmos de ajuste usando inferencia variacional y comparar los resultados con las implementaciones realizadas. Aunque existen múltiples estudios en los que comparan la eficiencia y rapidez de ambos algoritmos, como Luis Javier Rodríguez e Inés Torres [24]. También se podrían construir otros modelos para compararlos con los resultados obtenidos de este, y validar nuevamente la calidad del modelo programado.

También hemos comentado que en algunos casos el modelo queda sobre-ajustado a los datos y cuando trata de estimar valores nuevos comete un mayor error. Para evitar esto se pueden usar técnicas de validación cruzada progresiva vistas en la sección 3.4.4. También se pueden usar técnicas de regularización o ajustar los parámetros de la optimización en los casos de la biblioteca `TensorFlowProbability`.

Bibliografía

[1] Forbes Advisor. *Recession Definition: What Is A Recession?* https://www.forbes.com/advisor/investing/what-is-a-recession/. 2023 (vid. pág. 5).

[2] Leonard E Baum et al. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". En: *The Annals of Mathematical Statistics* 41.1 (1970), págs. 164-171. URL: https://www.jstor.org/stable/2239727 (vid. págs. 36, 39).

[3] Eli Bingham et al. "Pyro: Deep Universal Probabilistic Programming". En: *Journal of Machine Learning Research* (2018) (vid. pág. 5).

[4] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Ver. 0.3.13. 2018. URL: http://github.com/google/jax (vid. pág. 5).

[5] Jodie Buckby et al. "Model Checking for Hidden Markov Models". En: *Journal of Computational and Graphical Statistics* 29.4 (2020), págs. 859-874. DOI: 10.1080/10618600.2020.1743295. eprint: https://doi.org/10.1080/10618600.2020.1743295. URL: https://doi.org/10.1080/10618600.2020.1743295 (vid. pág. 65).

[6] Robert Galbraith. *The End of Economic Growth*. 2016. URL: https://www.theatlantic.com/business/archive/2016/11/economic-growth/506423/ (vid. pág. 55).

[7] Charles R. Harris et al. "Array programming with NumPy". En: *Nature* 585.7825 (sep. de 2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (vid. pág. 5).

[8] hmmlearn. *hmmlearn*. https://github.com/hmmlearn/hmmlearn. 2022 (vid. pág. 5).

[9] M A Hossain et al. "A comparative study between algorithms for time series forecasting on customer prediction: An investigation into the performance of ARIMA, RNN, LSTM, TCN and HMM". En: *arXiv preprint arXiv:1908.03533* (2019) (vid. pág. 4).

[10] John D Hunter. "Matplotlib: A 2D graphics environment". En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95 (vid. pág. 5).

[11] Investopedia. *Recession: What Is It and What Causes It*. https://www.investopedia.com/terms/r/recession.asp. 2023 (vid. pág. 5).

[12] Investopedia. *The 2007-2008 Financial Crisis in Review*. https://www.investopedia.com/articles/economics/09/financial-crisis-review.asp. 2009 (vid. pág. 6).

[13] Daniel Jurafsky y James H. Martin. *Speech and Language Processing*. MIT Press, 2023 (vid. pág. 35).

[14] Brian G Leroux y Martin L Puterman. "Maximum-penalized-likelihood estimation for independent and Markov-dependent mixture models". En: *Biometrics* 48.2 (1992), págs. 545-558 (vid. pág. 36).

[15] Heinz Linhart y Walter Zucchini. *Model Selection*. Wiley, 1986 (vid. pág. 50).

[16] R.J.A. Little y D.B. Rubin. *Statistical Analysis with Missing Data*. second. New York: Wiley, 2002 (vid. págs. 36, 37).

[17] Eric J. Ma. *Markov Models*. https://ericmjl.github.io/essays-on-data-science/machine-learning/markov-models/#autoregressive-emissions. 2022 (vid. pág. 77).

[18] Adam Paszke et al. "Automatic differentiation in PyTorch". En: *NIPS-W*. 2017 (vid. pág. 5).

[19] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830 (vid. pág. 5).

[20] Theodore M. Porter. *probability and statistics*. 2022. URL: https://www.britannica.com/science/probability (vid. pág. 3).

[21] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". En: *Proceedings of the IEEE* (1989) (vid. págs. 39, 78).

[22] Lawrence R Rabiner y Biing-Hwang JUANG. "The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models". En: *Proceedings of the IEEE* (1990) (vid. pág. 70).

[23] Christian P Robert y DM Titterington. "Reparameterization strategies for hidden Markov models and Bayesian approaches to maximum likelihood estimation". En: *Statistics and Computing* 8.2 (1998), págs. 145-158 (vid. pág. 36).

[24] Luis Javier Rodríguez e Inés Torres. "Comparative Study of the Baum-Welch and Viterbi Training Algorithms Applied to Read and Spontaneous Speech Recognition". En: *Pattern Recognition and Image Analysis*. IbPRIA, 2003, págs. 847-857 (vid. pág. 70).

[25] Jacob Schreiber. "Pomegranate: fast and flexible probabilistic modeling in python". En: *Journal of Machine Learning Research* 18.164 (2018), págs. 1-6 (vid. pág. 5).

[26] Soumya Shrivastava. *Cross Validation in Time Series*. https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4. 2020 (vid. págs. 46, 47).

[27] Emily Stewart. *What GDP does and doesn't tell us*. Jul. de 2022. URL: https://www.vox.com/the-goods/2022/7/28/23280969/gdp-report-recession-economy-climate-change-happiness (vid. págs. 5, 55).

[28] The pandas development team. *pandas-dev/pandas: Pandas*. Ver. latest. Feb. de 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134 (vid. pág. 5).

[29] TensorFlow. *Adam*. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam. 2023 (vid. págs. 5, 61).

[30] David Vere-Jones. *"Khinchin, Aleksandr Yakovlevich"*. *Encyclopedia of Statistical Sciences*. 2006 (vid. pág. 10).

[31] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". En: *Nature Methods* 17 (2020), págs. 261-272. DOI: 10.1038/s41592-019-0686-2 (vid. pág. 5).

[32] Andrew J Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". En: *IEEE Transactions on Information Theory* 13.2 (1973), págs. 260-269 (vid. pág. 34).

[33] Wikipedia. *Precision and recall — Wikipedia, The Free Encyclopedia*. 2023. URL: https://en.wikipedia.org/wiki/Precision_and_recall (vid. pág. 48).

[34] Nigel Wilson. "Redefining GDP And Economic Growth In Inclusive Capitalism Terms". En: *Forbes* (2020) (vid. págs. 5, 55).

[35] Aileen Zoo. *Practical Time Series Analysis*. O'Reilly Media, 2017 (vid. pág. 3).

[36] Walter Zucchini, Iain L. MacDonald y Roland Langrock. *Hidden Markov Models for Time Series An Introduction Using R*. 2.ª ed. CRC Press, 2016 (vid. págs. 11, 36, 38, 50).

Capítulo 6

Anexo

Para acceder al código es necesario acceder al siguiente enlace de github:

https://github.com/balalofernandez/FHMM

6.1. Demostraciones

Vamos a demostrar como la siguiente construcción de los vectores de probabilidades *forward* escalados cumplen con la ecuación 3.4.23:

$$\begin{aligned} \bar{\alpha}_1(i) &= \alpha_1(i) \\ \hat{\alpha}_1 &= \frac{\bar{\alpha}_1(i)}{\sum_i \bar{\alpha}_1(i)} = \hat{c}_1 \cdot \bar{\alpha}_1(i) \\ \bar{\alpha}_{t+1}(j) &= \sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j \left(O_{t+1}\right) \\ \hat{\alpha}_{t+1}(i) &= \frac{\bar{\alpha}_{t+1}(i)}{\sum_i \bar{\alpha}_{t+1}(i)} = \hat{c}_{t+1} \bar{\alpha}_{t+1}(i) \end{aligned} \tag{6.1.1}$$

Por inducción:

Caso base:

$$\bar{\alpha}_1(i) = \alpha_1(i), \quad \hat{\alpha}_1(i) = \frac{\alpha_1(i)}{\sum_j \alpha_1(j)}$$

satisface la ecuación (3.4.23) tomando $C_1 = \hat{c}_1$.

Paso de inducción: Suponiendo $\hat{\alpha}_t = C_t \alpha_t$, entonces

$$\begin{aligned}
\bar{\alpha}_{t+1}(j) &= C_t \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_j\left(O_{t+1}\right) \\
&= C_t \alpha_{t+1}(j) \\
\hat{c}_{t+1} &= \frac{1}{\sum \bar{\alpha}_{t+1}(i)} = \frac{1}{C_t \sum \alpha_{t+1}(j)} \\
\hat{\alpha}_{t+1}(i) &= \hat{c}_{t+1} \bar{\alpha}_{t+1}(i) \\
&= \frac{C_t \alpha_{t+1}(i)}{C_t \sum \alpha_{t+1}(i)} \\
&= \frac{\alpha_{t+1}(i)}{\sum \alpha_{t+1}(j)}
\end{aligned} \tag{6.1.2}$$

de la ecuación (6.1.2) y (3.4.23) podemos escribir una expresión de C_t en términos de $\hat{c}_t$:

$$\begin{aligned}
C_t &= \frac{1}{\hat{c}_{t+1} \sum \alpha_{t+1}(j)} = \frac{C_{t+1}}{\hat{c}_{t+1}} \\
C_t &= C_{t-1} \hat{c}_t = \prod_{\tau=1}^{t} \hat{c}_\tau
\end{aligned} \tag{6.1.3}$$

6.2. Modelos Ocultos de Markov Autorregresivos

El enfoque realizado en la sección anterior es en parte válido y sirve como introducción a los modelos ocultos de Markov. No obstante, el caso de estudio es una serie temporal, lo que significa que tenemos una variable de interés sobre la que observamos serie de valores en un orden cronológico. Además las variables tienen una correlación en el tiempo, es decir la variable $X^{(t)}$ en el tiempo t tendrá una correlación respecto de la variable $X^{(t-1)}$ del tiempo anterior $t-1$, o en general, dependerá de p estados anteriores. Esto se conoce como autocorrelación y el HMM anterior no recoge esta correlación pues asume que las observaciones son independientes.

Los procesos de autorregresión (AR) $X^{(t)}$ para $t \in T$ se escriben como:

$$x_t = \beta_0 + \beta_1 x_{t-1} + \beta_2 x_{t-2} + \cdots + \beta_p x_{t-p} + \epsilon_t$$

donde los coeficientes β_i, con $i = 1, ..., T$, son constantes y representan la dependencia de la variable $X^{(t)}$ en el instante actual respecto de los p estados anteriores. De aquí nace el término "auto", pues la propia variable X depende de su valor en un tiempo anterior. Usando la biblioteca `statsmodels` podemos observar que necesitaremos dos coeficientes para modelar la variación del PIB.

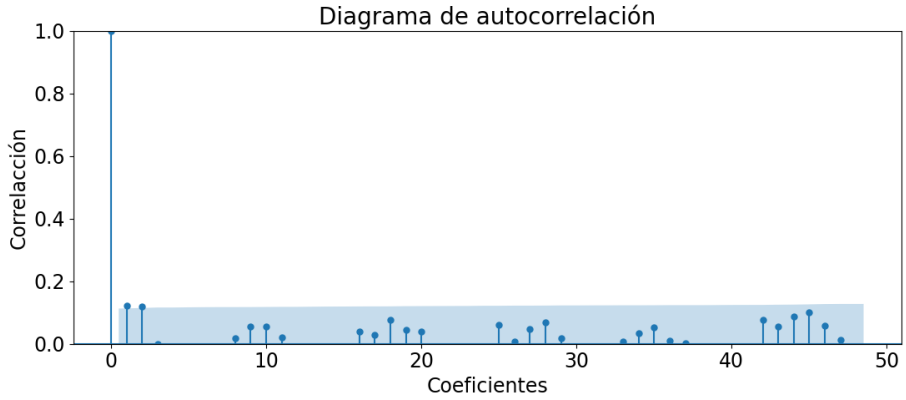


Figura 27: Diagrama de barras con los valores de β, la región azul indica los valores significativos, por lo que consideraremos únicamente dos coeficientes que aportan información significativa.

A su vez, existen modelos de autorregresión más complejos que incorporan medias móviles (ARMA) o los modelos de autorregresión integrados con media móvil (ARIMA) que intentan modelar procesos no estacionarios transformándolos a modelos estacionarios.

A la hora de trasladar este concepto a los modelos ocultos de Markov, tendremos que tener en cuenta que las observaciones dependen unas de otras y, además, puede existir una variación en los parámetros de las distribuciones de observación[1] a lo largo del tiempo.

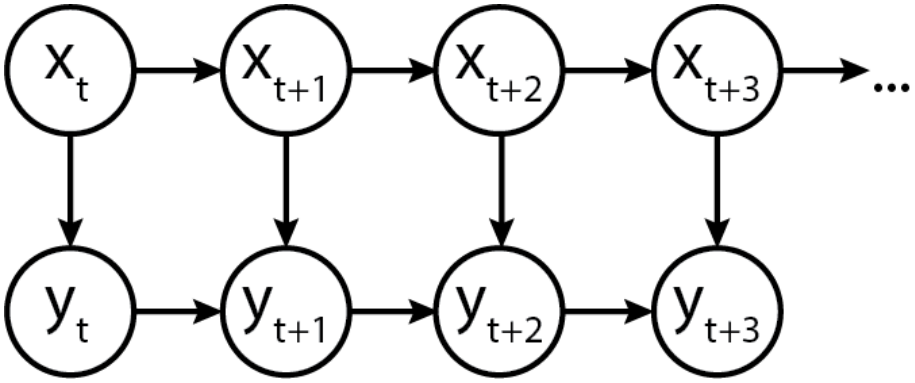


Figura 28: [17] Grafo ilustrando la nueva dependencia de los estados en los ArHMM. En este caso las observaciones dependen únicamente de la observación anterior, pero podrían depender de p observaciones anteriores.

De esta forma, la estructura autorregresiva recogerá la dependencia temporal de las observaciones, mientras que la cadena oculta de Markov recogerá cómo se transita entre los posibles estados. Este concepto de variación de los modelos ocultos de Markov se conoce como modelos ocultos de Markov con cambios de régimen o *Switching Markov Models*.

En cuanto al modelo de autorregresión que siguen las observaciones, se distinguen dos casos:

[1] También puede ocurrir una variación en las probabilidades de transición.

- Un modelo de autorregresión se considera **heterocedástico** si las variables del modelo presentan una varianza diferente en cada estado:

$$X^{(t)}|X^{(t-1)},...,X^{(t-p)},s_t \sim N(\boldsymbol{\beta}\cdot[x_{t-1},...,x_{t-p}]',\sigma_{s_t})$$

- Un modelo de autorregresión se considera **homocedástico** si las variables del modelo presentan la misma varianza en todos los estados:

$$X^{(t)}|X^{(t-1)},...,X^{(t-p)},s_t \sim N(\boldsymbol{\beta}\cdot[x_{t-1},...,x_{t-p}]',\sigma)$$

Para estos casos, si tenemos un gran número de observaciones (T es grande), la función de densidad será[21]:

$$f(\mathbf{X}) \simeq (\mathbf{2}\pi)^{-\frac{T}{2}} \exp\left\{-\frac{\mathbf{1}}{\mathbf{2}}\delta(\mathbf{X};\beta)\right\}$$

donde

$$\delta(X;\beta) = r_\beta(0)r(0) + 2\sum_{i=1}^{p} r_\beta(i)r(i)$$

$$\beta = [1,\beta_1,\beta_2,\ldots,\beta_p]'$$

$$r_\beta(i) = \sum_{n=0}^{p-i} \beta_n\beta_{n+i} \text{ con } \quad (\beta_0 = 1), 1 \leq i \leq p$$

$$r(i) = \sum_{n=0}^{T-i-1} Y_nY_{n+i} \quad 0 \leq i \leq p$$