

Detección de Cambios de Régimen en EUR/USD con Modelos HMM

1. Introducción

El objetivo de este estudio es aplicar **Modelos Ocultos de Markov (HMM)** para identificar regímenes de mercado en el par **EUR/USD intradía**.

Los mercados financieros presentan dinámicas no estacionarias: periodos de baja volatilidad intercalados con episodios de alta volatilidad o tendencias abruptas.

Un HMM permite modelar estos **estados ocultos** y estimar probabilísticamente en qué régimen se encuentra el mercado en cada momento.

1.2. Datos y preprocesado

- Fuente: **Dukascopy EUR/USD**, velas de 15 minutos, periodo **2000–2025**.
 - Variables originales: `open`, `high`, `low`, `close`, `volume`.
 - Features generados (`preprocessing.py`):
 - **Retornos**: `ret_log`.
 - **Rangos**: `range_hl`, `range_co`.
 - **Volumen**: `log_volume`.
 - **Indicadores técnicos**: `rsi_14`, `macd`, `macd_signal`, `adx`, `ma20_rel`, `mfi_14`.
 - **Volatilidad**: `atr_14`, `rolling_std_1h` ($\approx 1h$), `rolling_std_1d` (≈ 1 día).
 - **Tiempo**: `hour_sin`, `hour_cos`, `dow_sin`, `dow_cos`.
 - **Extremos recientes**: `dist_max20`, `dist_min20`.
 - Preprocesamiento:
 - (Opcional) **winsorización** de colas extremas.
 - **Estandarización z-score con media y desviación del TRAIN** de cada split → evita *data leakage*.
 - Splits:
 - **Por tamaño de ventana**: típicamente `train=200k`, `val=50k`, `test=50k` observaciones.
 - Opción de ventana deslizante con `step_size`.
-

2. Metodología: Modelado de Regímenes

2.1. El Problema y un Enfoque Inicial (Cadenas de Markov Simples)

El problema fundamental es que los mercados financieros no se comportan de manera constante. Usar una única distribución normal para modelar los rendimientos del EUR/USD es un error porque asume que la media y la volatilidad son fijas, lo cual no es cierto. Esto lleva a subestimar drásticamente la probabilidad de eventos extremos.

Solución: Una Distribución que Cambia en el Tiempo

La realidad es que la distribución que genera los rendimientos del mercado (la "distribución generadora de datos") cambia constantemente. En períodos de calma, la distribución puede ser estrecha, pero en períodos de crisis se vuelve mucho más ancha. El objetivo es crear un modelo donde la distribución de los rendimientos dependa del "régimen" o "estado" actual del mercado.

Clave del modelo: El Proceso de Variable Latente

Para lograr que nuestra distribución cambie, introducimos el concepto de una **variable latente**: un factor que no podemos observar directamente pero que influye en los datos que sí vemos.

En finanzas, la variable latente más evidente es la **volatilidad**. No podemos medirla directamente, pero podemos usar un *proxy* (por ejemplo, la desviación estándar de los retornos en una ventana móvil de 30 días). La idea es que el estado de esta volatilidad (baja, media, alta) determina la forma de la distribución de los rendimientos en ese momento.

- **Volatilidad baja**: distribución estrecha.
- **Volatilidad alta**: distribución mucho más ancha.

Proceso Latente (Volatilidad) -> Controla la Distribución Generadora de Datos -> Genera los Rendimientos Observados (EUR/USD)

Usando Cadenas de Markov Simples para modelar regímenes

Un primer enfoque consiste en modelar los regímenes usando este proceso latente de forma explícita:

1. **Definir Estados Manualmente**: Se simplifica el proceso de volatilidad (que es continuo) en un número finito de estados. Por ejemplo, 3 estados definidos por percentiles de la volatilidad histórica (ej. 33% más bajo es "Baja", 33%-66% es "Media", y >66% es "Alta").
2. **Calcular Probabilidades de Transición**: Se analiza la historia para ver con qué frecuencia el mercado pasa de un estado a otro (ej. de "Baja" a "Media"). Esto crea una matriz de transición.
3. **Ajustar Distribuciones**: Finalmente, para cada uno de los tres estados etiquetados, se ajusta una distribución normal diferente.
 - Alta Volatilidad tendrá una varianza mucho mayor.
 - Baja Volatilidad tendrá una varianza pequeña.

Este enfoque, aunque captura la dinámica, depende totalmente de nuestra elección del *proxy* (¿por qué volatilidad y no otra cosa?) y de cómo definimos los umbrales (¿por qué 33%?).

2.2. Enfoque Principal: Modelos Ocultos de Markov (HMM)

Ahora sí, hacemos una aproximación más compleja y robusta. En lugar de decirle al modelo cuándo es cada estado (como hicimos con los percentiles), **dejamos que el HMM aprenda los estados latentes automáticamente** a partir de los datos.

¿Cómo metemos el proceso latente con HMM?

La diferencia es clave:

- **Enfoque 1 (Cadenas de Markov Simples)**:

- Tú elegías el *proxy* para el proceso latente (ej. volatilidad histórica).
- Tú *definías* los estados (ej. "Baja", "Media", "Alta") basándote en ese proxy.

- **Enfoque 2 (HMM):**

- No definimos los estados manualmente. Simplemente **especificamos el número de estados latentes** (ej. 3 estados) y el tipo de distribución que creemos que tienen (ej. **Gaussiana**).
- El modelo, a través de sus algoritmos (Baum-Welch), **aprende automáticamente** cuáles son los parámetros óptimos (media y varianza/covarianza) de la distribución Gaussiana de **CADA** estado.
- Al mismo tiempo, aprende la **matriz de transición** entre esos estados.
- Los estados que "descubre" están basados en los patrones de todos los *features* que le proporcionamos (retornos, volumen, RSI, etc.), no solo en un proxy de volatilidad.

El hecho de usar un "**HMM Gaussiano**" (como haremos a continuación) no significa que el modelo final sea una distribución fija. Significa que cada *estado oculto individual* se modela como una distribución Gaussiana. El modelo general sigue siendo **dinámico**, ya que la distribución de los rendimientos en un día t es una **mezcla ponderada** de las distribuciones de cada estado, y esos pesos (las probabilidades de estar en cada estado) cambian constantemente.

2.3. HMM: Selección de Features y Evaluación (Fase 1 del Experimento)

Objetivo. Construir y evaluar subconjuntos de features (subsets) para el HMM Gaussiano que sean informativos para cambios de régimen y poco redundantes entre sí.

Evaluación de cada subconjunto/subset.

- **Antes de entrenar:** filtro intra-subset por correlación $|p| \geq 0.85$, eliminando la feature con menor puntuación de relevancia (no se toca el DataFrame global, solo el subset).
 - **Después de entrenar (con `experiment_hmm.py`):**
 - Métrica principal: `ll_val_per_obs_mean` (validación).
 - Penalización de complejidad: `AIC_mean`, `BIC_mean`.
 - Estabilidad/interpretabilidad (con `evaluate_hmm.py`):
 - Duraciones medianas por estado (evitar degeneración $L \approx 1$).
 - Matriz de transición con diagonales altas.
 - Separación por `var(ret_log)` entre estados (estado "volátil" bien diferenciado).
 - **Criterio final:** Elegimos los Top-K subsets por `ll_val_per_obs_mean`. Empates se resuelven con `BIC_mean` (menor es mejor) y sanidad de estados (duraciones razonables y separación clara).
-

2.4. HMM: Flujo de Selección Automática de Subsets

Métricas para decidir si un subset es "bueno"

- **Previo a entrenar:**
 - Relevancia media de sus features (ej. suma de MI).
 - Diversidad (baja correlación intra-subset).
- **Post-entrenar:**

- `ll_val_per_obs_mean` alto.
- AIC/BIC más bajos que otros.
- Estados con duraciones >2–3 barras y transiciones persistentes.
- Separación clara de volatilidad/retornos entre estados.

Flujo automático de selección de subsets


1. Calcular relevancia de cada feature

- Usa un proxy de régimen (volatilidad futura, retornos siguientes) y mide:
- Mutual Information (MI) o correlación absoluta con el proxy.
- Esto da un ranking global de features (las más "informativas" sobre dinámicas futuras arriba).
- *Ejemplo:* `rolling_std_1h` tendrá alta correlación con la volatilidad futura `rv_1h`. `rsi_14` puede correlacionarse con el retorno futuro. `log_volume` con ambos.

2. Generar candidatos de subsets

- Construyes automáticamente subsets de 3–5 features tomando las Top-N del ranking.
- Ejemplo: todas las combinaciones posibles de 3 features entre las 10 mejores (eso son 120 subsets).

3. Filtro de correlación intra-subset

- Para cada subset candidato:
- Calcula matriz de correlación entre sus features.
- Si algún par tiene $|p| > 0.85$, elimina la menos relevante (según ranking inicial).
- Si el subset se queda con  features → se descarta.
- Resultado: un conjunto de subsets filtrados y diversos.

4. Deduplicación y poda

- Quita duplicados.
- Si hay demasiados subsets, conserva solo los Top-M según un criterio rápido (ej. suma de MI de las features del subset).

5. Evaluación con el modelo

- Pasas todos esos subsets a `experiment_hmm.py`.
- Se entrenan los HMM y se comparan por `ll_val_per_obs_mean`, AIC/BIC, estabilidad de estados.

3. Cómo mejorar lo que tenemos hasta ahora con HMM

Hay que tener una idea clara, que a mí me confundía al principio;

Nuestro HMM es un modelo de UNA variable latente. Esa variable latente es el "estado" (Estado 1, Estado 2, Estado 3...). Es una única variable categórica que resume el régimen de mercado.

Tú no "añades más variables latentes" al HMM. Lo que haces es darle más features (información) para que su única variable latente sea más inteligente.

Entonces, cómo hacemos esta variable latente más inteligente?

- Si solo le damos el feature de `ret_log`, su decisión será pobre.
- Si le damos informes de Volatilidad, Momentum y Volumen (tus features), su estado de "Pánico" será mucho más preciso.

Nuestro objetivo es comprimir la información de diferentes "velocidades" del mercado en features densos y no correlacionados.

El mercado tiene memoria a diferentes plazos. Un estado de "pánico" no se define solo por la volatilidad de la última hora, sino por una combinación de volatilidad alta a corto, medio y largo plazo.

- Primero, crearemos familias de features que midan el mismo concepto (ej. volatilidad) pero en diferentes ventanas de tiempo:
 - Familia de Volatilidad
(`rolling_std_1h`, `rolling_std_4h`(volatilidad intradía), `rolling_std_1d` (volatilidad diaria), `atr_14_1h`, `atr_14_4h`)...
 - Familia de Momentum
(`rsi_14`, `rsi_56`, `macd`, `ma_diff_1h_vs_4h`)
 - Familia de Volumen
(`log_volume_1h`, `log_volume_4h`, `log_volume_1d`)
- Después reducimos dimensionalidad (con PCA por ejemplo)
El problema es que muchos de estos features están altamente correlacionados (`rolling_std_1h` y `rolling_std_4h` se moverán juntas), un HMM Gaussiano sufre con features colineales (hace que la matriz de covarianza sea inestable), entonces lo que vamos a hacer es aplicar PCA a cada familia. Entonces nos quedamos con PC1 de cada familia, así tenemos `VOLATILITY_FACTOR`, `MOMENTUM_FACTOR`, `VOLUME_FACTOR`, esto de cada conjunto de features.

Ahora simplemente entrenamos el HMM con estos features 🤖

Otra mejora, Cambiar de distribución

Ahora mismo estamos usando un Gaussian HMM.

Esto asume que los rendimientos dentro de un mismo régimen (ej. "Calma") siguen una distribución normal. Esto sigue siendo una simplificación. vaya, que incluso cuando diga "calma" puede haber un evento atípico.

Una mejora que se suele implementar es cambiar la distribución gaussiana por una t-student.

- Captura los grandes cambios de régimen (de Calma a Pánico) con los estados de Markov.
- Captura los fat tails dentro de cada régimen con la distribución T-Student
*** fat tails son eventos que la distribución dice que ocurren con muy poca frecuencia cuando en realidad ocurre mucho más frecuentemente ***