# Investigación sobre el ML en el criptotrading.

## Areas de aplicación del ML al criptotrading:

* Predicción de precios: Los algoritmos de machine learning pueden entrenarse para analizar datos históricos de precios y otros indicadores relacionados con las criptomonedas, y luego realizar predicciones sobre los movimientos futuros de los precios. Esto puede ayudar a los traders a tomar decisiones más informadas sobre cuándo comprar o vender criptomonedas.
* Análisis de sentimiento: El machine learning también se utiliza para analizar el sentimiento del mercado en las redes sociales, foros y otras fuentes de datos en línea. Estos análisis de sentimiento pueden proporcionar información sobre las opiniones y expectativas de los inversores en relación con ciertas criptomonedas, lo que puede ser útil para predecir movimientos de precios y tomar decisiones comerciales.
* Detección de anomalías: Los algoritmos de machine learning pueden ser entrenados para identificar patrones y anomalías inusuales en los datos del mercado de criptomonedas. Esto puede ayudar a detectar posibles eventos de manipulación del mercado, hackeos u otras actividades sospechosas que podrían afectar los precios de las criptomonedas.
* Gestión de riesgos: El machine learning se utiliza en el criptotrading para desarrollar modelos de gestión de riesgos. Estos modelos pueden evaluar la volatilidad del mercado, la liquidez, el tamaño de las operaciones y otros factores para determinar estrategias de gestión de riesgos óptimas, como la asignación de capital y el establecimiento de stop-loss.

De las areas anteriores, podemos ordenarlas por dificultad, de la más sencilla a la más compleja:

1. Predicción de Precios.
2. Detección de Anomalías.
3. Gestión de Riesgos
4. Análisis de Sentimiento.
5. Reconocimiento de Patrones.
6. Análisis de Hiperparametros.

A mi parecer, es una buena manera de proceder el desarrollo de la Predicción de Precios primeramente por obvias razones, y a la par se puede tratar el desarrollo de la Gestión de Riesgos y el desarrollo de la Detección de Anomalías. Aunque la Gestión de Riesgos me parece más importante.  
  
Pienso que el Análisis de Sentimiento debería de desarrollarse una vez que tengamos un MVP (hasta que haya ganancias). Esto porque el desarrollo supone un costo mayor al de otras áreas ya que se necesitan GPUs o TPUs de servicios Cloud.  
Aunque esto lo podemos pensar mejor porque podríamos hacer análisis de sentimiento con mi máquina a manera de prueba, lo que sería ideal antes de comenzar a pagar por un servicio ya que tendríamos mejor idea de cómo utilizar los recursos, además de que podría ser este análisis el más efectivo en la predicción de precios. Sin embargo, esto también sería un costo extra.

El área de reconocimiento de patrones se puede aplicar en diversas áreas, tal como el análisis de hiperparámetros.

Por último menciono el Análisis de Hiperparámetros puesto que es algo que se debe o se puede implementar desde el área de la predicción de precios, así como a las otras áreas. Pero sería bueno hablar si lo implementamos solo básicamente al principio, porque puede ser costoso en recursos.

## 1.- ML en la Predicción de Precios para Criptotrading:

En la predicción de precios para el criptotrading, el machine learning se utiliza para analizar datos históricos de precios y otros indicadores relacionados con las criptomonedas, con el objetivo de predecir los movimientos futuros de los precios. Aquí hay algunas técnicas comunes utilizadas en este campo:

* Regresión: La regresión es una técnica de machine learning que se utiliza para predecir valores numéricos, como el precio futuro de una criptomoneda. Se pueden aplicar diferentes algoritmos de regresión, como regresión lineal, regresión polinómica o regresión de vectores de soporte (SVM), para encontrar la relación entre los datos históricos y los precios futuros.
* Redes neuronales: Las redes neuronales son modelos de machine learning inspirados en la estructura y funcionamiento del cerebro humano. En la predicción de precios de criptomonedas, se pueden utilizar redes neuronales para aprender patrones complejos en los datos históricos y hacer predicciones basadas en esas relaciones aprendidas. Las redes neuronales pueden tener diferentes arquitecturas, como redes neuronales feedforward, redes neuronales recurrentes (RNN) o redes neuronales convolucionales (CNN).
* Modelos de series temporales: Los modelos de series temporales son específicos para el análisis y la predicción de datos secuenciales en el tiempo. En el criptotrading, los precios de las criptomonedas suelen ser datos de series temporales. Algunos modelos populares para el análisis de series temporales son el modelo autorregresivo integrado de media móvil (ARIMA), el modelo de promedio móvil autoregresivo (ARMA) y el modelo autorregresivo de media móvil (ARMA).
* Aprendizaje reforzado: El aprendizaje reforzado es otra técnica utilizada en la predicción de precios para el criptotrading. En este enfoque, un agente de aprendizaje interactúa con el entorno de trading y aprende a tomar decisiones de compra y venta basadas en recompensas recibidas. El agente utiliza algoritmos de aprendizaje reforzado, como Q-learning o algoritmos genéticos, para mejorar su rendimiento a lo largo del tiempo.

A como lo veo, la mejor forma de proceder es: desarrollar la técnica de Regresión primeramente y aplicarla a la testnet de binance, posteriormente, desarrollar Modelos de Series Temporales, lo que también implementaríamos a la testnet. Es buena idea probar por separado ambas herramientas y también tratar de unirlas en una sola predicción.

A partir de aquí también va a ser necesario un script que nos permita monitorear en tiempo real las compras, las ventas y los beneficios o pérdidas... Esto claro para los resultados de la testnet primeramente.

Después del desarrollo de estas herramientas, será una buena idea agregar redes neuronales a los datos obtenidos por las herramientas anteriores. Y más adelante, podría ser agregado un Modelo de Aprendizaje Reforzado. Y bueno, se pueden buscar otras herramientas.

La transición entre la testnet y la red real será poco a poco, pero solo hasta que se haya confirmado un porcentaje de trades ganados superior al 50-55% de manera constante en la testnet (aunque puede ser revisado el porcentaje mínimo aceptado), y además será necesario tener en operación, al menos, la predicción de precios y la gestión de riesgos. Podríamos intentar trabajar en la red real como una prueba con solo la predicción de precios, pero eso sería solo un experimento para ir haciendo pruebas en la red real y tener experiencia en ello.

Un buen modelo podría replicar los resultados de la testnet con una varianza “aceptable”, pero no es aceptable un rendimiento negativo en la red real.

### 1.1.- Técnicas de ML aplicables en el área de la predicción de precios:

Todas las técnicas que voy a mencionar en las secciones siguientes están sujetas a ser usadas o no, debemos de hacer una investigación más a profundidad de cada técnica y con dicha información tomar la decisión de cuáles van a ser las técnicas que vamos a utilizar al comienzo. Las que no se utilicen al comienzo pueden ser integradas más adelante si vemos que son necesarias, pero por eso es necesario una mejor comprensión de cada técnica para poder tomar decisiones adecuadas con base en los datos que tenemos.

Para cada técnica habrá que crear un nuevo documento en donde se detalle su funcionamiento, sus variantes, y a qué datos se suele aplicar, entre otras cosas que se puedan encontrar en la investigación.

Antes de comenzar con las descripciones de cada técnica, me gustaría presentar la siguiente tabla

| **Orden de Uso en Criptotrading** | **Mis Recomendaciones para Criptotrading** |
| --- | --- |
| Regresión Lineal | Regresión de Vectores de Soporte (SVM) |
| Regresión Polinómica | Redes Neuronales |
| Regresión de Vectores de Soporte (SVM) | Regresión Polinómica |
| Redes Neuronales | Regresión Lineal |
| ARIMA | ARIMA |
| ARMA | ARMA |
| Aprendizaje Reforzado | Aprendizaje Reforzado |

En donde podemos ver las técnicas más usadas en el criptotrading del lado derecho por orden de mayor uso, y en el izquierdo las técnicas recomendadas por chatgpt en orden de importancia para él.

Cómo vemos, son justamente las herramientas de ML descritas anteriormente para la predicción de precios.

Según chatgpt, esta es una posible forma de armar un sistema de predicción de precios para el criptotrading:

*“Si estás armando un sistema de Predicción de Precios para el criptotrading, aquí hay una posible forma de combinar diferentes técnicas de regresión utilizando el enfoque de ensamble y aprovechando las fortalezas de cada una:*

*1. Recopilación y preprocesamiento de datos: Comienza recopilando datos históricos de precios de criptomonedas y otros indicadores relevantes, como volumen de operaciones, indicadores técnicos, noticias relacionadas, etc. Asegúrate de limpiar y preprocesar los datos para eliminar valores atípicos y manejar cualquier problema de calidad de datos.*

*2. Técnicas de regresión individuales: Aplica varias técnicas de regresión, como regresión lineal, regresión polinómica, regresión de vectores de soporte (SVM), redes neuronales, ARIMA, ARMA, entre otras. Cada técnica puede generar su propia predicción de precio.*

*3. Ensamble de modelos: Utiliza un enfoque de ensamble para combinar las predicciones generadas por diferentes técnicas de regresión. Puedes aplicar técnicas como el promedio de las predicciones o utilizar métodos más sofisticados como el ensamble de votación o el ensamble apilado (stacking), donde se entrena un modelo adicional para combinar las predicciones de los modelos base.*

*4. Ajuste y optimización: Realiza ajustes y optimizaciones en los hiperparámetros de cada modelo individual y del ensamble. Esto implica realizar ajustes finos en cada técnica de regresión para mejorar su rendimiento individual y también ajustar la forma en que se combinan las predicciones en el ensamble.*

*5. Evaluación y validación: Utiliza datos de prueba o técnicas de validación cruzada para evaluar el rendimiento de tu sistema de predicción de precios. Puedes utilizar métricas como el error cuadrático medio (MSE) o la raíz del error cuadrático medio (RMSE) para comparar el rendimiento de los modelos individuales y del ensamble.*

*En cuanto a la aplicación de técnicas de ML a los resultados de otras, puede ser beneficioso si las técnicas son complementarias y abordan diferentes aspectos del problema de predicción de precios. Por ejemplo, puedes utilizar la salida de una técnica de regresión como características de entrada para otro modelo de machine learning, como una red neuronal. Esto puede ayudar a capturar relaciones no lineales más complejas y mejorar la precisión de las predicciones.*

*Sin embargo, es importante tener en cuenta que agregar más complejidad al sistema no siempre garantiza mejores resultados. Es crucial realizar pruebas y validaciones adecuadas para determinar si la incorporación de técnicas adicionales realmente mejora el rendimiento del sistema. Además, es fundamental tener en cuenta el riesgo y la volatilidad asociados con el criptotrading, y no confiar únicamente en los resultados de la predicción para tomar decisiones comerciales, sino también considerar otros factores y realizar análisis adicionales.”*

Ahora sí, veamos las técnicas que mencionamos anteriormente una a una. Comenzaré describiendo las regresiones, después los modelos de series temporales, luego las redes neuronales y al final el aprendizaje reforzado.

#### 1.1.1.- Regresiones:

La regresión es un enfoque de aprendizaje automático supervisado utilizado para predecir valores numéricos basados en la relación entre variables de entrada y salida. En el contexto del criptotrading, se utiliza para predecir los precios futuros de las criptomonedas en función de datos históricos y otros indicadores relevantes.

Existen varios algoritmos de regresión que se pueden aplicar en la predicción de precios de criptomonedas, y cada uno tiene sus propias características. Aquí hay algunos ejemplos:

1.- Regresión lineal: La regresión lineal es uno de los métodos más simples y ampliamente utilizados en la predicción de precios. Busca establecer una relación lineal entre las variables de entrada y salida. El modelo intenta ajustar una línea recta que mejor se ajuste a los datos históricos y luego utiliza esa línea para predecir los precios futuros. Sin embargo, la regresión lineal puede no ser suficientemente flexible para capturar relaciones no lineales entre los datos.

2.- Regresión polinómica: La regresión polinómica es una extensión de la regresión lineal que permite modelar relaciones no lineales. En lugar de ajustar una línea recta, se ajusta un polinomio de grado superior a los datos históricos. Esto permite capturar relaciones más complejas entre las variables de entrada y salida. Sin embargo, es importante tener en cuenta que el uso de un grado polinomial muy alto puede llevar al sobreajuste del modelo.

3.- Regresión de vectores de soporte (SVM): Los vectores de soporte son algoritmos de aprendizaje automático utilizados tanto para clasificación como para regresión. En el caso de la predicción de precios, se utiliza la regresión de vectores de soporte (SVR). Este enfoque busca encontrar una función que se ajuste a los datos históricos con un margen de error aceptable. El modelo SVM busca maximizar el margen entre los datos y la función de regresión, lo que lo hace menos sensible a valores atípicos.

Es importante tener en cuenta que estos algoritmos de regresión requieren datos históricos adecuados y relevantes para entrenar y evaluar el modelo. Además, el rendimiento del modelo puede verse afectado por la calidad de los datos, la elección de características relevantes y otros factores. Es recomendable realizar una exploración exhaustiva de los datos, aplicar técnicas de preprocesamiento adecuadas y ajustar los hiperparámetros del modelo para obtener los mejores resultados posibles en la predicción de precios para el criptotrading.

Podemos hacer entrenamiento de la regresión polinómica y la regresión de vectores de soporte en el largo plazo y entrenar la lineal en el corto plazo, debido a la naturaleza de los datos, los cuales son no lineales, pero pueden llegar a aproximarse mejor linealmente en periodos cortos. Sería pensar si en realidad vale la pena hacer una investigación al respecto sobre combinar de alguna manera las predicciones de largo plazo de la polinómica y la SVM y las de corto plazo de la lineal y tomar una decisión. Porque en realidad podría no valer la pena el tiempo implementado si por separado nos dan resultados suficientemente buenos.

Aquí entra la técnica de *Ensamble de modelos* que obtuvimos de chatgpt. Así como también la técnica de *Ajuste y optimización*, la cual es básicamente un Análisis de Hiperparámetros, el cual debe ser automatizado por obvias razones y puede aplicarse en diferentes estadios del desarrollo.

Señale solo 3 métodos (en amarillo), que son los que vamos a ver, sin embargo, podemos investigar cualquiera de los siguientes métodos para aplicarlo:

* Regresión lineal
* Regresión polinómica
* Regresión logística
* Regresión de Ridge
* Regresión de Lasso
* Regresión elástica net
* Regresión de mínimos cuadrados parciales (PLS)
* Regresión de mínimos cuadrados ordinarios (OLS)
* Regresión de Huber
* Regresión de Theil-Sen
* Regresión de cuantiles
* Regresión robusta
* Regresión bayesiana
* Regresión no paramétrica
* Regresión por vecinos más cercanos (KNN Regression)
* Regresión por árboles de decisión
* Regresión por bosques aleatorios (Random Forest Regression)
* Regresión por máquinas de soporte vectorial (SVM Regression)
* Regresión por redes neuronales
* Regresión por modelos de ensamblaje (ensemble models)

Los que marqué en azul turquesa son las regresiones que me parecen buena opción para probarlas junto con las otras 3.

##### 1.1.1.1.- Regresión lineal:

La regresión lineal es un método de aprendizaje automático supervisado utilizado para predecir o modelar la relación entre una variable dependiente continua y una o más variables independientes, que pueden ser continuas o categóricas. El objetivo es encontrar una función lineal que mejor se ajuste a los datos observados y pueda usarse para hacer predicciones sobre nuevos datos.

El modelo de regresión lineal asume que la relación entre la variable dependiente y las variables independientes puede aproximarse mediante una combinación lineal de los coeficientes de las variables independientes. La forma general de un modelo de regresión lineal simple es:

y = β0 + β1\*x1 + β2\*x2 + ... + βn\*xn + ε

Donde:

- y es la variable dependiente que queremos predecir.

- β0 es el término de intersección o sesgo (intercept).

- β1, β2, ..., βn son los coeficientes o pesos de las variables independientes x1, x2, ..., xn.

- x1, x2, ..., xn son las variables independientes.

- ε es el término de error, que captura la diferencia entre el valor predicho y el valor real de y.

La regresión lineal se basa en encontrar los mejores valores para los coeficientes β0, β1, β2, ..., βn que minimicen la suma de los cuadrados de los errores (método de mínimos cuadrados). Esto se puede lograr utilizando técnicas de optimización como la minimización de la función de costo o mediante la aplicación de métodos matriciales.

Una vez que se han estimado los coeficientes, se puede utilizar el modelo de regresión lineal para hacer predicciones sobre nuevas observaciones. Simplemente se sustituyen los valores de las variables independientes en la ecuación del modelo para obtener el valor estimado de la variable dependiente.

Es importante tener en cuenta que la regresión lineal asume una relación lineal entre las variables independientes y la variable dependiente. Si la relación es no lineal, se pueden aplicar transformaciones a las variables para capturar mejor la relación no lineal o se pueden explorar otros métodos de regresión más flexibles.

Además, es fundamental evaluar la calidad del modelo de regresión lineal. Se utilizan diversas métricas de evaluación, como el coeficiente de determinación (R^2), el error cuadrático medio (MSE) o la raíz del error cuadrático medio (RMSE), para determinar qué tan bien se ajusta el modelo a los datos observados y qué tan bien generaliza a nuevos datos.

La regresión lineal es un método ampliamente utilizado debido a su simplicidad y facilidad de interpretación. Sin embargo, su rendimiento puede ser limitado en situaciones donde la relación entre las variables es no lineal o hay fuertes violaciones de los supuestos de regresión lineal, como la presencia de valores atípicos o la falta de linealidad en los residuos. En tales casos, se pueden explorar técnicas más avanzadas, como las regresiones polinómicas, regresiones de máxima verosimilitud o técnicas de aprendizaje automático más complejas.

Un pequeño código en python con la librería sklearn:

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Cargar los datos de criptotrading

data = pd.read\_csv('datos\_criptotrading.csv') # Asegúrate de reemplazar 'datos\_criptotrading.csv' con tu archivo de datos

# Dividir los datos en variables independientes (X) y variable dependiente (y)

X = data[['Variable1', 'Variable2', 'Variable3']] # Reemplaza 'Variable1', 'Variable2', 'Variable3' con los nombres de tus variables independientes

y = data['Precio'] # Reemplaza 'Precio' con el nombre de tu variable dependiente

# Dividir los datos en conjuntos de entrenamiento y prueba

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Crear el modelo de regresión lineal

model = LinearRegression()

# Entrenar el modelo utilizando los datos de entrenamiento

model.fit(X\_train, y\_train)

# Realizar predicciones en el conjunto de prueba

y\_pred = model.predict(X\_test)

# Calcular el error cuadrático medio (MSE)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Error cuadrático medio (MSE): {mse}')

# Obtener los coeficientes y el término de intersección

coefficients = model.coef\_

intercept = model.intercept\_

print('Coeficientes:', coefficients)

print('Término de intersección:', intercept)

Aún falta probarlo y adaptarlo a nuestros datos.

##### 1.1.1.2.- Regresión Polinómica:

Al respecto de la regresión polinómica podemos decir lo siguiente:

La regresión polinómica es una extensión del modelo de regresión lineal en el cual se permite ajustar relaciones no lineales entre la variable dependiente y las variables independientes. En lugar de ajustar una línea recta, la regresión polinómica utiliza una función polinómica para modelar los datos.

La forma general de un modelo de regresión polinómica de grado "n" es:

y = β0 + β1x + β2x^2 + ... + βn\*x^n + ε

Donde:

y es la variable dependiente que queremos predecir.

β0, β1, β2, ..., βn son los coeficientes o pesos de los términos polinómicos.

x es la variable independiente.

x^2, x^3, ..., x^n son las potencias de x hasta el grado n.

ε es el término de error.

La regresión polinómica busca encontrar los mejores valores para los coeficientes β0, β1, β2, ..., βn que minimicen la suma de los cuadrados de los errores, de manera similar a la regresión lineal.

La principal ventaja de la regresión polinómica es que permite capturar relaciones no lineales entre las variables, lo que puede ser especialmente útil cuando los datos muestran patrones curvilíneos o no se ajustan bien a una línea recta. Al agregar términos polinómicos de grado superior, el modelo puede ajustarse mejor a los datos y proporcionar predicciones más precisas.

Sin embargo, es importante tener en cuenta que la regresión polinómica también tiene algunas limitaciones. A medida que aumenta el grado del polinomio, el modelo se vuelve más complejo y puede llevar al sobreajuste (overfitting), lo que significa que el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. Por lo tanto, es fundamental realizar una validación adecuada y considerar técnicas de regularización, como la regularización de Ridge o LASSO, para controlar la complejidad del modelo y evitar el sobreajuste.

Es importante hacer hincapie en las técnicas de regularización con el objetivo de mejorar el modelo.

Un código sencillo de este método es el siguiente:

import numpy as np

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

# Datos de ejemplo

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.array([2, 3, 6, 10, 15])

# Crear características polinómicas de grado 2

poly\_features = PolynomialFeatures(degree=2)

X\_poly = poly\_features.fit\_transform(X)

# Crear el modelo de regresión lineal

model = LinearRegression()

# Entrenar el modelo utilizando las características polinómicas

model.fit(X\_poly, y)

# Realizar predicciones

X\_test = np.array([6]).reshape(-1, 1)

X\_test\_poly = poly\_features.transform(X\_test)

y\_pred = model.predict(X\_test\_poly)

print(f'Predicción: {y\_pred}')

Básicamente se utiliza el método de la regresión lineal pero se le añaden características polinomiales.

##### 1.1.1.3.- Regresión de Vectores de Soporte:

Ahora veamos más sobre la última técnica, de la cual encontramos que:

La Regresión de Vectores de Soporte (SVR, por sus siglas en inglés) es una técnica de aprendizaje automático utilizada para problemas de regresión, donde el objetivo es predecir un valor numérico continuo en lugar de clasificar en clases discretas. SVR es una variante del algoritmo SVM (Support Vector Machine) adaptada para problemas de regresión.

A diferencia de la regresión lineal, que busca ajustar una línea o un hiperplano a los datos, la SVR busca encontrar una función no lineal que se ajuste a los datos de manera óptima. Utiliza los vectores de soporte, que son puntos de datos cercanos a la función de regresión, para definir la función objetivo.

El objetivo principal de la SVR es minimizar el error de regresión y, al mismo tiempo, controlar la complejidad del modelo para evitar el sobreajuste. La complejidad del modelo se controla mediante parámetros como C y ε, que permiten establecer un equilibrio entre la precisión del ajuste y la flexibilidad del modelo.

El proceso básico para aplicar SVR es el siguiente:

1. Preparación de los datos: Al igual que en cualquier problema de regresión, es necesario preparar los datos de entrada y salida en un formato adecuado.

2. Selección del kernel: La SVR utiliza el "kernel trick", que permite mapear los datos a un espacio de características de mayor dimensión para encontrar hiperplanos no lineales de separación. Los kernels más comunes utilizados en SVR son el kernel lineal, el kernel polinómico y el kernel radial (RBF).

3. Definición de la función de pérdida: La función de pérdida en SVR tiene como objetivo minimizar el error de regresión. La forma específica de la función de pérdida depende del tipo de SVR utilizado, como SVR lineal, SVR no lineal, SVR con kernel polinómico, etc.

4. Optimización: La optimización en SVR implica encontrar los vectores de soporte y los coeficientes correspondientes que minimicen la función de pérdida. Esto se puede lograr utilizando técnicas de optimización como el descenso de gradiente o métodos de programación cuadrática.

5. Predicción: Una vez que se ha entrenado el modelo SVR, se puede utilizar para realizar predicciones en nuevos datos de entrada y estimar los valores continuos correspondientes.

Es importante destacar que la SVR tiene en cuenta los vectores de soporte, que son los puntos de datos más relevantes y cercanos a la función de regresión. Estos puntos desempeñan un papel crítico en la definición del modelo y tienen un impacto directo en la precisión de las predicciones.

La implementación de SVR se puede realizar utilizando bibliotecas populares de aprendizaje automático en Python, como scikit-learn, que proporciona una implementación flexible y fácil de usar de SVR. La biblioteca scikit-learn ofrece una variedad de opciones y parámetros para ajustar y personalizar el modelo SVR según los requisitos del problema de regresión específico.

Un código sencillo en python con sklearn sería el siguiente:

import numpy as np

from sklearn.svm import SVR

# Datos de entrenamiento

X\_train = np.array([[1], [2], [3], [4], [5]]) # Características

y\_train = np.array([2, 4, 6, 8, 10]) # Etiquetas

# Crear el modelo SVR

svr\_model = SVR(kernel='linear', C=1.0, epsilon=0.2)

# Entrenar el modelo

svr\_model.fit(X\_train, y\_train)

# Datos de prueba

X\_test = np.array([[6], [7], [8]]) # Características

# Realizar predicciones

y\_pred = svr\_model.predict(X\_test)

# Imprimir las predicciones

print("Predicciones:")

for i in range(len(X\_test)):

print(f"Característica: {X\_test[i][0]}, Predicción: {y\_pred[i]}")

En este ejemplo, se utiliza una implementación de SVR con un kernel lineal. Se crea una instancia de la clase SVR de scikit-learn y se especifican los parámetros kernel='linear' para utilizar un kernel lineal, C=1.0 para el parámetro de regularización y epsilon=0.2 para controlar el margen de error permitido.

Luego, se entrena el modelo utilizando los datos de entrenamiento (X\_train y y\_train), y se realiza la predicción en nuevos datos de prueba (X\_test). Finalmente, se imprimen las predicciones correspondientes.

Es importante tener en cuenta que este es solo un ejemplo básico para ilustrar cómo se utiliza la técnica SVR. En la práctica, es posible ajustar los parámetros, explorar diferentes kernels y realizar una validación cruzada para encontrar el modelo óptimo para el problema de regresión específico.

#### 1.1.2.- Modelos de Series Temporales:

Los modelos de series temporales son técnicas de inteligencia artificial (AI) ampliamente utilizadas en el análisis y la predicción de datos secuenciales, como los precios de criptomonedas en el criptotrading. Estos modelos se centran en la estructura temporal de los datos y buscan capturar patrones, tendencias y estacionalidad para realizar predicciones futuras.

A continuación, describiré algunas de las técnicas de modelos de series temporales que se aplican comúnmente al criptotrading:

1.- Modelos ARIMA (Autoregressive Integrated Moving Average): Los modelos ARIMA son una clase de modelos estadísticos que se utilizan para predecir valores en series temporales estacionarias. Estos modelos consideran la autocorrelación y utilizan una combinación de componentes autoregresivos (AR), de promedio móvil (MA) y de integración (I) para modelar la tendencia, la estacionalidad y los errores. Los modelos ARIMA son flexibles y se pueden ajustar a diferentes patrones temporales.

2.- Modelos de suavizamiento exponencial: Los modelos de suavizamiento exponencial son métodos que se basan en ponderar los valores pasados para realizar una predicción. Estos modelos son adecuados para series temporales con tendencia y estacionalidad. Algunas variantes populares de los modelos de suavizamiento exponencial incluyen el suavizamiento exponencial simple (SES), el suavizamiento exponencial doble (Holt) y el suavizamiento exponencial triple (Holt-Winters).

3.- Redes Neuronales Recurrentes (RNN): Las RNN son un tipo de arquitectura de redes neuronales especialmente diseñadas para modelar secuencias de datos, como las series temporales. Estas redes tienen conexiones recurrentes que les permiten recordar información anterior y utilizarla en la predicción futura. Las variantes populares de las RNN incluyen las Long Short-Term Memory (LSTM) y las Gated Recurrent Unit (GRU), que han demostrado ser efectivas en la predicción de series temporales debido a su capacidad para capturar dependencias a largo plazo.

4.- Modelos de Estado Espacial (State Space Models): Los modelos de estado espacial son técnicas estadísticas que representan una serie temporal como una combinación de componentes observables y no observables. Estos modelos se utilizan para capturar la tendencia, la estacionalidad y otros factores que pueden influir en la serie temporal. Los modelos de estado espacial, como el filtro de Kalman y el suavizador de Kalman, se utilizan comúnmente en la predicción de series temporales.

5.- Modelos de Aprendizaje Profundo (Deep Learning): Los modelos de aprendizaje profundo, como las redes neuronales convolucionales (CNN) y las redes neuronales generativas (GAN), también se aplican en la predicción de series temporales en el criptotrading. Estos modelos utilizan capas y conexiones más profundas que las redes neuronales tradicionales, lo que les permite aprender representaciones más complejas de los datos y capturar patrones temporales más sofisticados.

Es importante señalar aquí que el punto 3 y el 5 son redes neuronales, por lo que no las vamos a describir en esta sección. Solo vamos a dejar la información puesto que se relaciona con la aplicación de redes neuronales a las series de tiempo, sin embargo, abordaremos 5 puntos, los otros 2 son los siguientes:

6.- GARCH (Generalized Autoregressive Conditional Heteroskedasticity):

GARCH es un modelo estadístico utilizado para modelar y predecir la volatilidad en las series temporales financieras. Este modelo se basa en la idea de que la varianza de una serie temporal puede cambiar con el tiempo y estar influenciada por los valores pasados de la serie. El modelo GARCH captura tanto la autocorrelación en los residuos como la heteroscedasticidad condicional, lo que lo hace adecuado para el modelado de la volatilidad en el criptotrading y otros mercados financieros.

7.- Prophet:

Prophet es una librería de código abierto desarrollada por Facebook para el análisis y la predicción de series temporales. Utiliza un enfoque aditivo basado en descomposición de tendencia y estacionalidad. Prophet puede manejar automáticamente la detección de cambios en las tendencias, la identificación de componentes estacionales y la gestión de valores atípicos en los datos. Esta herramienta es especialmente popular en el campo del criptotrading debido a su facilidad de uso y capacidad para modelar series temporales con patrones complejos y no lineales.

Dentro de los modelos de series temporales tenemos la siguiente lista:

* ARIMA (Autoregressive Integrated Moving Average)
* SARIMA (Seasonal Autoregressive Integrated Moving Average)
* GARCH (Generalized Autoregressive Conditional Heteroskedasticity)
* VAR (Vector Autoregression)
* LSTM (Long Short-Term Memory)
* GRU (Gated Recurrent Unit)
* RNN (Recurrent Neural Network)
* Prophet
* Wavelet transform
* Holt-Winters
* Kalman filter
* Exponential Smoothing
* Fourier Transform
* Seasonal Decomposition of Time Series (STL)
* TBATS (Trigonometric Exponential Smoothing State Space Model)
* State Space Models

En color amarillo están las que abordaremos y en morado son las redes neuronales que se proponen como modelos de series temporales puesto que también son útiles a la hora de modelar series temporales. Las redes neuronales las abordaremos en la siguiente sección.

Ahora pasemos a la descripción de los métodos que obtuvimos:

##### 1.1.2.1.- ARIMA:

ARIMA (Autoregressive Integrated Moving Average) es un modelo estadístico utilizado para analizar y predecir series temporales. El modelo ARIMA combina componentes de autocorrelación, integración y medias móviles para modelar la estructura temporal de los datos.

El acrónimo ARIMA representa los siguientes componentes:

1. AR (Autoregressive): El componente autoregresivo se refiere a la dependencia de una observación actual de valores pasados de la misma serie temporal. En un modelo ARIMA, el término AR indica el número de observaciones pasadas que se utilizarán en la predicción actual.

2. I (Integrated): El componente integrado se refiere a la diferenciación de la serie temporal para hacerla estacionaria. La estacionariedad implica que las propiedades estadísticas de la serie, como la media y la varianza, no cambian a lo largo del tiempo. La diferenciación implica tomar la diferencia entre las observaciones sucesivas para eliminar las tendencias o patrones no estacionarios.

3. MA (Moving Average): El componente de media móvil utiliza los errores de predicción pasados para modelar la influencia de los residuos en la predicción actual. Al igual que el componente autoregresivo, el término MA indica el número de errores de predicción pasados que se tendrán en cuenta en la predicción actual.

El modelo ARIMA se ajusta a los datos utilizando métodos estadísticos como la estimación de máxima verosimilitud. Una vez ajustado, se puede utilizar para hacer predicciones futuras en la serie temporal.

ARIMA es una técnica ampliamente utilizada en el análisis de series temporales, incluido el criptotrading. Sin embargo, la selección adecuada de los parámetros AR, I y MA requiere un análisis cuidadoso de los datos y puede implicar pruebas y ajustes para obtener un modelo óptimo.

Un código sencillo para aplicar este método es el siguiente:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

# Cargar los datos

data = pd.read\_csv('datos.csv', index\_col='fecha', parse\_dates=True)

series = data['precio']

# Ajustar el modelo ARIMA

model = ARIMA(series, order=(p, d, q))

model\_fit = model.fit()

# Obtener las predicciones

predictions = model\_fit.predict(start='2023-01-01', end='2023-12-31')

# Visualizar los resultados

plt.plot(series, label='Datos reales')

plt.plot(predictions, color='red', label='Predicciones')

plt.legend()

plt.xlabel('Fecha')

plt.ylabel('Precio')

plt.title('Modelo ARIMA - Predicciones de precios')

plt.show()

En este código, debes reemplazar 'datos.csv' con el nombre y la ubicación de tu archivo de datos. También debes ajustar los valores de 'p', 'd' y 'q' en la línea order=(p, d, q) para especificar los órdenes del modelo ARIMA que deseas utilizar.

Una vez que el modelo está ajustado, puedes obtener las predicciones llamando al método predict y especificando el rango de fechas para el cual deseas hacer las predicciones.

Aun no se ha probado el código.

##### 1.1.2.2.- SARIMA:

SARIMA (Seasonal Autoregressive Integrated Moving Average) es una extensión del modelo ARIMA que se utiliza para modelar y predecir series temporales que exhiben tanto componentes de estacionalidad como de tendencia. Al igual que ARIMA, el modelo SARIMA combina componentes de autocorrelación, integración y medias móviles, pero también incorpora un componente de estacionalidad para capturar patrones repetitivos a lo largo del tiempo.

El acrónimo SARIMA representa los siguientes componentes:

1. AR (Autoregressive): El componente autoregresivo captura la dependencia de una observación actual de los valores pasados de la misma serie temporal, teniendo en cuenta tanto la estacionalidad como la tendencia.

2. I (Integrated): El componente integrado se refiere a la diferenciación de la serie temporal para hacerla estacionaria, similar a ARIMA. La diferencia con ARIMA es que SARIMA también considera la diferenciación estacional para eliminar patrones estacionales.

3. MA (Moving Average): El componente de media móvil modela la influencia de los errores de predicción pasados en la predicción actual, teniendo en cuenta tanto la estacionalidad como la tendencia.

4. S (Seasonal): El componente estacional se refiere a la incorporación de patrones estacionales en el modelo. Esto implica la inclusión de términos autoregresivos, diferencias y medias móviles en un período estacional.

El modelo SARIMA se ajusta a los datos utilizando métodos estadísticos similares a ARIMA. Los parámetros clave que se deben especificar al ajustar un modelo SARIMA son: el orden del componente AR, I y MA (representados como p, d, q), y el orden del componente estacional (representado como P, D, Q, s).

SARIMA es especialmente útil cuando se trabaja con datos que exhiben patrones estacionales, como en el caso de muchas series temporales financieras, incluido el criptotrading. Al modelar tanto la estacionalidad como la tendencia, SARIMA puede proporcionar predicciones más precisas y capturar mejor las fluctuaciones a largo plazo en los datos.

Un pequeño código de esta técnica sería el siguiente:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX

# Cargar los datos

data = pd.read\_csv('datos.csv', index\_col='fecha', parse\_dates=True)

series = data['precio']

# Ajustar el modelo SARIMA

model = SARIMAX(series, order=(p, d, q), seasonal\_order=(P, D, Q, s))

model\_fit = model.fit()

# Obtener las predicciones

predictions = model\_fit.predict(start='2023-01-01', end='2023-12-31')

# Visualizar los resultados

plt.plot(series, label='Datos reales')

plt.plot(predictions, color='red', label='Predicciones')

plt.legend()

plt.xlabel('Fecha')

plt.ylabel('Precio')

plt.title('Modelo SARIMA - Predicciones de precios')

plt.show()

Al igual que en el código del modelo ARIMA, debes reemplazar 'datos.csv' con el nombre y la ubicación de tu archivo de datos. Además, debes ajustar los valores de 'p', 'd', 'q' para el componente ARIMA y los valores de 'P', 'D', 'Q', 's' para el componente estacional.

Una vez que el modelo SARIMA está ajustado, puedes obtener las predicciones llamando al método predict y especificando el rango de fechas para el cual deseas hacer las predicciones.

Tampoco este código se ha probado.

##### 1.1.2.3.- Exponential Smoothing:

Exponential Smoothing (Suavizado Exponencial) es una técnica utilizada para predecir y suavizar series temporales, especialmente aquellas que no presentan patrones estacionales claros. Esta técnica se basa en la idea de que los valores más recientes tienen más peso o importancia en la predicción que los valores anteriores.

El suavizado exponencial calcula la predicción futura como una combinación ponderada de los valores pasados de la serie temporal. La ponderación se realiza mediante la asignación de pesos exponenciales decrecientes a los valores pasados, donde los valores más recientes tienen un peso mayor.

Existen diferentes variantes del suavizado exponencial, cada una con diferentes características y suposiciones. Aquí se describen tres de las variantes más comunes:

1. Simple Exponential Smoothing (Suavizado Exponencial Simple): En esta variante, la predicción futura se calcula como una combinación ponderada de los valores pasados de la serie temporal, donde todos los valores tienen el mismo peso. La fórmula general es:

Predicción futura = Peso \* Valor actual + (1 - Peso) \* Predicción anterior

El peso es un parámetro que determina la importancia relativa de los valores pasados y puede ser ajustado para optimizar la predicción.

2. Double Exponential Smoothing (Suavizado Exponencial Doble): Esta variante se utiliza para series temporales que también exhiben una tendencia en los datos. Además de considerar los valores pasados, también se incorpora una estimación de la tendencia en la predicción. Se utiliza una fórmula similar a la del suavizado exponencial simple, pero con dos pesos diferentes para los valores pasados y la estimación de la tendencia.

3. Triple Exponential Smoothing (Suavizado Exponencial Triple o Holt-Winters): Esta variante se utiliza para series temporales que también presentan una componente estacional. Además de considerar los valores pasados y la tendencia, también se incorpora una estimación de la estacionalidad en la predicción. La fórmula general incluye tres pesos diferentes para los valores pasados, la tendencia y la estacionalidad.

El suavizado exponencial es una técnica sencilla pero efectiva que puede proporcionar predicciones suaves y estables para series temporales. Sin embargo, no es adecuada para todas las situaciones y puede no capturar patrones complejos en los datos. Es importante tener en cuenta las características de los datos y las suposiciones de cada variante de suavizado exponencial al aplicar esta técnica.

El Double Exponential Smoothing (Suavizado Exponencial Doble) es una técnica de suavizado utilizada para predecir y suavizar series temporales que presentan una tendencia en los datos. Es una extensión del Simple Exponential Smoothing que incorpora una estimación de la tendencia en la predicción.

Aquí vemos que tenemos 3 variantes del suavizado exponencial. A primera vista, podemos ver que la suavización exponencial doble y la triple son los métodos que más nos conviene utilizar, puesto que las series de trading pueden contener tendencia y estacionalidad.

###### 1.1.2.3.1.- Double Exponential Smoothing:

El Double Exponential Smoothing utiliza dos componentes principales: nivel y tendencia. El componente de nivel representa el nivel base de la serie temporal, mientras que el componente de tendencia representa la dirección y la velocidad de cambio de la serie temporal.

La fórmula general para calcular la predicción en Double Exponential Smoothing es la siguiente:

Predicción futura = Nivel actual + Tendencia actual

El nivel y la tendencia se actualizan en cada paso de tiempo utilizando las siguientes ecuaciones:

Nivel actual = (1 - Alfa) \* (Valor actual) + Alfa \* (Nivel anterior + Tendencia anterior)

Tendencia actual = (1 - Beta) \* (Nivel actual - Nivel anterior) + Beta \* Tendencia anterior

Donde:

- Valor actual: El valor actual de la serie temporal.

- Nivel anterior: El nivel en el paso de tiempo anterior.

- Tendencia anterior: La tendencia en el paso de tiempo anterior.

- Alfa: El factor de suavizado para el nivel, que determina el peso dado al valor actual en la actualización del nivel. Un valor más alto de Alfa da más peso al valor actual y hace que la serie se ajuste más rápidamente a los cambios.

- Beta: El factor de suavizado para la tendencia, que determina el peso dado a la diferencia entre los niveles en la actualización de la tendencia. Un valor más alto de Beta hace que la tendencia se ajuste más rápidamente a los cambios.

El Double Exponential Smoothing es útil cuando se desea predecir y suavizar una serie temporal que presenta una tendencia lineal. Sin embargo, no es adecuado para capturar cambios no lineales o patrones más complejos en los datos. En tales casos, pueden ser necesarias técnicas más avanzadas, como el Triple Exponential Smoothing (Holt-Winters).

Es importante ajustar adecuadamente los parámetros Alfa y Beta para obtener las mejores predicciones. Esto puede hacerse mediante métodos de optimización, como la minimización del error cuadrado medio o mediante validación cruzada en diferentes conjuntos de datos.

De acuerdo con lo que encontré, esta técnica funciona bien para comportamientos lineales, sin embargo, no es buena para capturar comportamientos no lineales. Recomienda la técnica de Holt-Winters para casos no lineales.

###### 1.1.2.3.2.- Triple Exponential Smoothing or Holt-Winters:

Holt-Winters es una técnica de suavizado exponencial triple utilizada para modelar y predecir series temporales que exhiben componentes de tendencia y estacionalidad. Esta técnica es una extensión del suavizado exponencial doble (Double Exponential Smoothing) al que se agrega un componente adicional para capturar patrones estacionales en los datos.

El método de Holt-Winters tiene tres componentes principales: nivel (level), tendencia (trend) y estacionalidad (seasonality). Cada componente se actualiza en función de los valores observados y los valores estimados en el paso de tiempo anterior. Estos componentes se calculan utilizando fórmulas recursivas y se combinan para producir las predicciones futuras.

Existen dos variantes principales del método Holt-Winters: aditiva y multiplicativa, que se eligen en función de la naturaleza de la estacionalidad en los datos.

1. Holt-Winters Aditivo:

En esta variante, se asume que la estacionalidad tiene un efecto aditivo constante en la serie temporal. Los pasos para calcular las predicciones son los siguientes:

- Actualización del nivel (level):

Nivel actual = Alfa \* (Valor actual - Estacionalidad anterior) + (1 - Alfa) \* (Nivel anterior + Tendencia anterior)

- Actualización de la tendencia (trend):

Tendencia actual = Beta \* (Nivel actual - Nivel anterior) + (1 - Beta) \* Tendencia anterior

- Actualización de la estacionalidad (seasonality):

Estacionalidad actual = Gamma \* (Valor actual - Nivel actual) + (1 - Gamma) \* Estacionalidad anterior

- Predicción futura:

Predicción futura = Nivel actual + Tendencia actual + Estacionalidad actual

2. Holt-Winters Multiplicativo:

En esta variante, se asume que la estacionalidad tiene un efecto multiplicativo constante en la serie temporal. Los pasos para calcular las predicciones son similares a los de Holt-Winters aditivo, pero se utiliza una multiplicación en lugar de una suma en la actualización del nivel y la predicción futura.

El método de Holt-Winters es particularmente útil cuando se trabaja con series temporales que muestran tanto una tendencia como una estacionalidad, como los datos financieros y económicos. Permite capturar y modelar estos patrones, proporcionando predicciones más precisas.

Al aplicar el método de Holt-Winters, es necesario ajustar los parámetros Alfa, Beta y Gamma para obtener las mejores predicciones. Esto se puede lograr utilizando técnicas de optimización o validación cruzada en diferentes conjuntos de datos. Además, es importante tener en cuenta que el método Holt-Winters asume que los patrones de tendencia y estacionalidad son estables a lo largo del tiempo y puede no ser adecuado para series con cambios bruscos o patrones irregulares.

Y tomando en cuenta nuestros datos, los cuales pueden contener cambios bruscos y patrones irregulares, podemos decir que incluso esta técnica no es una buena aproximación a los datos de criptoactivos. Sin embargo podríamos probarla.

##### 1.1.2.4.- State Space Models:

State Space Models (Modelos de Espacio de Estados), también conocidos como modelos de espacio de estados ocultos (Hidden Markov Models), son una clase de modelos estadísticos utilizados para describir y predecir series temporales. Estos modelos son particularmente útiles cuando se trabaja con datos que exhiben una estructura dinámica compleja y están sujetos a incertidumbre.

En un State Space Model, se considera que una serie temporal es el resultado de dos procesos diferentes:

1. Proceso oculto (Hidden process): Representa la dinámica subyacente o no observada de la serie temporal. Este proceso se modela mediante una secuencia de estados ocultos que evolucionan a lo largo del tiempo. Cada estado oculto representa una combinación de factores que determina el comportamiento de la serie temporal.

2. Proceso observable (Observable process): Representa las observaciones o mediciones realizadas en el tiempo. Estas observaciones están relacionadas con los estados ocultos a través de una función de observación que puede ser determinística o probabilística.

Un State Space Model se compone de dos ecuaciones principales:

1. Ecuación de estado (State equation): Describe cómo evolucionan los estados ocultos a lo largo del tiempo. Puede ser un sistema de ecuaciones lineales o no lineales que relaciona el estado en el tiempo t con el estado en el tiempo t-1 y posiblemente con otras variables externas.

2. Ecuación de observación (Observation equation): Relaciona las observaciones en el tiempo t con los estados ocultos en el mismo tiempo t. Esta ecuación puede ser lineal o no lineal y puede contener términos de ruido o error.

La estimación de los parámetros en un State Space Model generalmente se realiza mediante técnicas de inferencia estadística, como el filtro de Kalman o el algoritmo de Expectation-Maximization (EM). Estas técnicas permiten estimar los estados ocultos y los parámetros del modelo en función de las observaciones disponibles.

Los State Space Models son ampliamente utilizados en diferentes aplicaciones, como pronóstico de series temporales, detección de anomalías, modelado de sistemas dinámicos y reconocimiento de patrones. Son especialmente útiles cuando se trabaja con datos que presentan estructuras complejas y cambios en el tiempo, lo que los hace aplicables en áreas como finanzas, economía, procesamiento de señales y muchas otras.

Bajo estas condiciones, es esta técnica la que mejor se ajusta a la naturaleza de nuestros datos. Además de ello, también se puede aplicar en el área de detección de anomalías y en reconocimiento de patrones. Lo que nos permitiría reutilizar el código.

Algunos modelos de este tipo son los siguientes:

1.- Modelo de espacio de estados con volatilidad estocástica (Stochastic Volatility State Space Model): Este modelo se utiliza para modelar la volatilidad de los precios en el criptotrading. Incorpora un componente oculto que representa la volatilidad, que puede variar a lo largo del tiempo de manera estocástica. Este modelo ayuda a capturar la naturaleza cambiante de la volatilidad en los mercados de criptomonedas.

2.- Modelo de espacio de estados con cambio de régimen (Switching State Space Model): Este modelo se utiliza para identificar y modelar diferentes regímenes o estados en los precios de las criptomonedas. Puede ayudar a capturar cambios abruptos en la dinámica de los precios y ajustar el modelo en consecuencia.

3.- Modelo de espacio de estados con componentes de tendencia y estacionalidad (State Space Model with Trend and Seasonality): Este modelo se utiliza para modelar y predecir la tendencia y la estacionalidad en los precios de las criptomonedas. Puede incorporar componentes de tendencia lineal o no lineal, así como patrones estacionales recurrentes, lo que permite capturar los patrones cíclicos y de largo plazo en los precios.

4.- Modelo de espacio de estados con saltos (Jump State Space Model): Este modelo se utiliza para modelar eventos de salto o cambios bruscos en los precios de las criptomonedas. Los saltos pueden ser causados por eventos importantes como noticias, anuncios regulatorios o cambios en la percepción del mercado. Este modelo ayuda a capturar y modelar estos eventos de manera efectiva.

5.- Modelo de espacio de estados con información externa (State Space Model with External Information): Este modelo se utiliza para incorporar información externa relevante en el análisis de criptomonedas, como datos económicos, noticias o indicadores financieros. Puede ayudar a mejorar la precisión de las predicciones al aprovechar información adicional que pueda afectar los precios de las criptomonedas.

6.- Modelo de espacio de estados de volatilidad heterocedástica condicional (GARCH): El modelo GARCH es utilizado para modelar la volatilidad de los precios en el criptotrading. A diferencia de los modelos de volatilidad estocástica mencionados anteriormente, el GARCH asume que la volatilidad es condicionalmente heterocedástica, lo que significa que puede variar con el tiempo y estar correlacionada con los errores pasados.

7.- Modelo de espacio de estados con cambio de nivel (Level Shift State Space Model): Este modelo se utiliza para detectar y modelar cambios de nivel en los precios de las criptomonedas. Puede ser útil cuando hay eventos significativos que provocan cambios permanentes en los precios, como cambios en la política regulatoria o anuncios importantes.

8.- Modelo de espacio de estados con cambio de pendiente (Slope Change State Space Model): Este modelo se utiliza para detectar y modelar cambios en la pendiente o la tendencia de los precios de las criptomonedas. Puede ser útil para identificar cambios en la dirección de la tendencia y ajustar el modelo en consecuencia.

9.- Modelo de espacio de estados de cambio de régimen oculto (Hidden Regime Switching State Space Model): Este modelo es similar al modelo de cambio de régimen mencionado anteriormente, pero incorpora un proceso oculto adicional que indica el régimen o estado actual del mercado. Puede ser útil para capturar cambios no observables en los precios y ajustar el modelo de manera adecuada.

Estos son solo algunos ejemplos de modelos de espacio de estados aplicados al criptotrading.

Ahora veamos una tabla de ellos ordenados por importancia y por más usados:

| **Más utilizados en criptotrading** | **Mayor potencial de ganancias** |
| --- | --- |
| Modelo de espacio de estados lineal (Linear State Space Model) | Modelo de espacio de estados con cambio de régimen (Switching State Space Model) |
| Modelo de espacio de estados no lineal (Nonlinear State Space Model) | Modelo de espacio de estados con volatilidad estocástica (Stochastic Volatility State Space Model) |
| Modelo de espacio de estados con cambio de régimen (Switching State Space Model) | Modelo de espacio de estados no lineal (Nonlinear State Space Model) |
| Modelo de espacio de estados con volatilidad estocástica (Stochastic Volatility State Space Model) | Modelo de espacio de estados con cambio de nivel (Level Shift State Space Model) |
| Modelo de espacio de estados con cambio de nivel (Level Shift State Space Model) | Modelo de espacio de estados lineal (Linear State Space Model) |
| Modelo de espacio de estados con cambio de pendiente (Slope Change State Space Model) | Modelo de espacio de estados con cambio de pendiente (Slope Change State Space Model) |
| Modelo de espacio de estados de cambio de régimen oculto (Hidden Regime Switching State Space Model) | Modelo de espacio de estados de cambio de régimen oculto (Hidden Regime Switching State Space Model) |
| Modelo de espacio de estados con ruido aditivo (State Space Model with Additive Noise) | Modelo de espacio de estados con ruido multiplicativo (State Space Model with Multiplicative Noise) |
| Modelo de espacio de estados con ruido multiplicativo (State Space Model with Multiplicative Noise) | Modelo de espacio de estados con ruido aditivo (State Space Model with Additive Noise) |

Va a ser importante enfocarnos en esta técnica puesto que se ve prometedora con respecto a la descripción que se dio más arriba.

Aquí me gustaría parar para decir lo siguiente:

Veo un poco a las regresiones como un nivel básico de predicción, el cual muy probablemente no sea suficiente para obtener grandes ganancias pero sí tal vez un retorno moderado como el que se espera de un banco o instituciones similares, o quizá incluso pérdidas si no se desarrollan bien.

Por su parte, veo mayor potencial de retorno en los modelos de series temporales, los cuales hasta ahora han sido más complejos, como podemos comparar de las descripciones de ambas técnicas. Esto porque claramente abordan mejor la naturaleza de nuestros datos que las regresiones.

Y aunque no hemos llegado a esa sección, es muy probable que las redes neuronales tengan más potencial de retorno que las dos técnicas anteriores.

Sin embargo, lo que yo pienso más va a ser rentable será la combinación adecuada de herramientas de estas 3 técnicas. Siendo posible agregar una red neuronal a los resultados de las regresiones y las series temporales. Aunque puede que no mejore mucho la rentabilidad, pero ya lo veremos poco a poco con el desarrollo paulatino de estas áreas.

Ahora veamos algunos de los modelos mencionados en la tabla:

###### 1.1.2.4.1.- Modelo de Espacio de Estados Lineal:

El Modelo de Espacio de Estados Lineal (Linear State Space Model) es un enfoque estadístico utilizado para modelar series temporales y otras secuencias de datos. Se basa en la idea de descomponer la serie temporal en un estado no observado (oculto) y un proceso de observación.

En este modelo, se asume que el sistema está gobernado por un conjunto de ecuaciones lineales. El estado no observado evoluciona en el tiempo de acuerdo con una ecuación de transición, mientras que las observaciones se generan a partir del estado oculto a través de una ecuación de observación. Estas ecuaciones están definidas por parámetros que se pueden estimar a partir de los datos.

La ecuación de transición describe cómo evoluciona el estado no observado a lo largo del tiempo. Puede haber una relación lineal entre el estado en el tiempo actual y el estado en el tiempo anterior, así como la influencia de otros factores. Esta ecuación puede tener en cuenta cambios de nivel, tendencia, estacionalidad u otros patrones que se observen en los datos.

La ecuación de observación describe cómo las observaciones se generan a partir del estado oculto. Generalmente, se asume una relación lineal entre el estado y las observaciones, pero también pueden incorporarse otros factores como ruido o errores.

Una vez que se ha definido el modelo de espacio de estados lineal, se utiliza un algoritmo de filtrado, como el filtro de Kalman, para estimar los estados ocultos en cada punto de tiempo y obtener estimaciones de las variables de interés.

Este modelo es útil en el criptotrading ya que permite capturar patrones lineales en los datos y ajustarse a diferentes condiciones del mercado. Se puede utilizar para realizar predicciones, detección de anomalías y análisis de tendencias en los precios de las criptomonedas.

Es importante destacar que el Modelo de Espacio de Estados Lineal asume linealidad en las relaciones y puede no ser adecuado si los datos presentan patrones no lineales.

Aquí cabe recalcar que este modelo puede no ser de utilidad en periodos largos, pero puede funcionar en periodos cortos de tiempo para ofrecernos una aproximación más ajustada a lo real en el corto plazo. Lo cual podría combinarse bien con los siguientes modelos que incorporan otros factores que representan mejor a la naturaleza de nuestros datos.

El código de este método sería el siguiente:

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace import tools

from statsmodels.tsa.statespace import representation

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados lineal

class LinearStateSpaceModel(tools.FrozenRepresentation, representation.FrozenRepresentation):

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace import tools

from statsmodels.tsa.statespace import representation

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados lineal

class LinearStateSpaceModel(tools.FrozenRepresentation, representation.FrozenRepresentation):

def \_\_init\_\_(self, endog):

super().\_\_init\_\_(endog, k\_states=1, k\_posdef=1, initialization='known')

def update(self, params, \*\*kwargs):

self['design', 0, 0] = params[0]

self['transition', 0, 0] = params[1]

self['selection', 0, 0] = params[2]

self['state\_cov', 0, 0] = params[3]

# Ajustar el modelo a los datos

model = LinearStateSpaceModel(y)

results = model.fit()

# Realizar predicciones

n\_forecast = 50

forecast = results.forecast(steps=n\_forecast)

# Plotear los datos y las predicciones

plt.figure(figsize=(10, 6))

plt.plot(t, y, label='Datos')

plt.plot(np.arange(n, n + n\_forecast), forecast, label='Predicción')

plt.legend()

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.title('Modelo de Espacio de Estados Lineal')

plt.show()

En este ejemplo, se generan datos sintéticos utilizando un proceso AR(1). Luego, se define una clase LinearStateSpaceModel que hereda de las clases FrozenRepresentation y FrozenRepresentation de statsmodels. Esta clase representa el modelo de espacio de estados lineal y define los parámetros del modelo (diseño, transición, selección y covarianza del estado). A continuación, se ajusta el modelo a los datos utilizando el método fit() y se realizan predicciones para los próximos 50 pasos de tiempo utilizando el método forecast(). Por último, se plotean los datos y las predicciones.

El cual no se ha probado aún.

###### 1.1.2.4.2.- Modelo de Espacio de Estados con Cambio de Régimen:

El Modelo de Espacio de Estados con Cambio de Régimen (Switching State Space Model) es una técnica utilizada en el análisis de series temporales, incluyendo el criptotrading, para capturar cambios bruscos y no lineales en los datos. Este modelo permite identificar y modelar diferentes regímenes o estados en los que los datos pueden encontrarse, y ajustar los parámetros del modelo en consecuencia.

En este modelo, se asume que los datos observados están gobernados por un conjunto de ecuaciones de espacio de estados, donde el sistema se divide en múltiples regímenes. Cada régimen tiene su propio conjunto de ecuaciones y parámetros, lo que refleja diferentes dinámicas o comportamientos en los datos.

El modelo utiliza una variable latente o oculta llamada "estado de régimen" que indica en qué régimen se encuentra el sistema en cada período de tiempo. Esta variable puede tomar diferentes valores discretos que representan los diferentes regímenes. La transición entre los regímenes puede ser gobernada por un proceso estocástico, como una cadena de Markov, donde la probabilidad de cambio de régimen depende del régimen actual.

El Modelo de Espacio de Estados con Cambio de Régimen se estima mediante técnicas de estimación de máxima verosimilitud, donde se busca encontrar los parámetros óptimos del modelo que maximicen la probabilidad conjunta de los datos observados y los estados ocultos. La estimación de los parámetros se realiza a través de algoritmos de filtrado y suavizado, como el filtro de Kalman.

Una vez estimado el modelo, se puede utilizar para realizar predicciones y análisis. El modelo permite capturar cambios no lineales en los datos y adaptarse a diferentes condiciones del mercado. Por ejemplo, en el criptotrading, puede ayudar a identificar períodos de alta volatilidad o tendencias abruptas en los precios y ajustar las estrategias de trading en consecuencia.

Es importante destacar que la aplicación del Modelo de Espacio de Estados con Cambio de Régimen requiere un análisis cuidadoso de los datos y una selección adecuada de los regímenes y variables relevantes. Además, el rendimiento del modelo puede depender de la calidad de los datos y de las suposiciones realizadas. Por lo tanto, es recomendable realizar una evaluación rigurosa y considerar otras técnicas de modelado antes de utilizar este modelo en el criptotrading.

Un código sencillo, con la librería pykalman, sería el siguiente:

import numpy as np

import matplotlib.pyplot as plt

from pykalman import KalmanFilter

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados con cambio de régimen

transition\_matrices = [[0.8, 0], [0, 0.9]]

transition\_covariance = np.eye(2)

observation\_matrices = np.array([[1, 0], [1, 0]])

observation\_covariance = np.eye(2)

initial\_state\_mean = [0, 0]

initial\_state\_covariance = np.eye(2)

kf = KalmanFilter(transition\_matrices=transition\_matrices,

transition\_covariance=transition\_covariance,

observation\_matrices=observation\_matrices,

observation\_covariance=observation\_covariance,

initial\_state\_mean=initial\_state\_mean,

initial\_state\_covariance=initial\_state\_covariance)

# Ajustar el modelo a los datos

filtered\_state\_means, filtered\_state\_covariances = kf.filter(y)

# Realizar predicciones

n\_forecast = 50

filtered\_state\_means\_forecast, filtered\_state\_covariances\_forecast = kf.filter(y, n\_steps\_ahead=n\_forecast)

# Obtener las predicciones y los intervalos de confianza

forecast\_means = filtered\_state\_means\_forecast[:, 0]

forecast\_std = np.sqrt(filtered\_state\_covariances\_forecast[:, 0, 0])

forecast\_ci\_upper = forecast\_means + 1.96 \* forecast\_std

forecast\_ci\_lower = forecast\_means - 1.96 \* forecast\_std

# Plotear los datos y las predicciones

plt.figure(figsize=(10, 6))

plt.plot(t, y, label='Datos')

plt.plot(np.arange(n, n + n\_forecast), forecast\_means, label='Predicción')

plt.fill\_between(np.arange(n, n + n\_forecast), forecast\_ci\_lower, forecast\_ci\_upper, color='gray', alpha=0.3, label='Intervalo de Confianza')

plt.legend()

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.title('Modelo de Espacio de Estados con Cambio de Régimen')

plt.show()

En este ejemplo, se generan datos sintéticos utilizando un proceso AR(1). Luego, se define el modelo de espacio de estados con cambio de régimen utilizando los parámetros de transición y covarianza, matrices de observación y los valores iniciales. Se utiliza la clase KalmanFilter de la librería pykalman para construir el modelo. A continuación, se ajusta el modelo a los datos utilizando el método filter(), lo cual proporciona los estados filtrados (mean) y las covarianzas correspondientes. Se realizan predicciones utilizando el método filter() con el argumento n\_steps\_ahead para especificar la cantidad de pasos futuros a predecir. Finalmente, se plotean los datos originales, las predicciones y los intervalos de confianza.

Ahora veamos un código con la librería statsmodels:

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados con cambio de régimen

model = sm.tsa.MarkovRegression(y, k\_regimes=2)

# Ajustar el modelo a los datos

result = model.fit()

# Obtener la estimación de los estados latentes

filtered\_states = result.smoothed\_state

# Plotear los datos y los estados latentes

plt.figure(figsize=(10, 6))

plt.plot(t, y, label='Datos')

plt.plot(t, filtered\_states[:, 0], label='Estado 1')

plt.plot(t, filtered\_states[:, 1], label='Estado 2')

plt.legend()

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.title('Modelo de Espacio de Estados con Cambio de Régimen')

plt.show()

En este ejemplo, se generan datos sintéticos utilizando un proceso AR(1). Luego, se utiliza la clase MarkovRegression de statsmodels.tsa para definir el modelo de espacio de estados con cambio de régimen. El argumento k\_regimes=2 especifica que el modelo tiene dos regímenes. A continuación, se ajusta el modelo a los datos utilizando el método fit(). Se obtiene la estimación de los estados latentes utilizando el atributo smoothed\_state del resultado del ajuste. Finalmente, se plotean los datos originales y los estados latentes a lo largo del tiempo.

Los códigos no se han probado.

###### 1.1.2.4.3.- Modelo de Espacio de Estados con Volatilidad Estocástica:

El Modelo de Espacio de Estados con Volatilidad Estocástica (Stochastic Volatility State Space Model en inglés) es un enfoque utilizado para modelar y estimar la volatilidad de una serie de datos a lo largo del tiempo. Este modelo es particularmente útil en el análisis financiero, incluido el criptotrading, donde la volatilidad es una medida esencial para evaluar los riesgos y las oportunidades.

En este modelo, se asume que la volatilidad de la serie de datos cambia con el tiempo y está influenciada por un proceso estocástico. La volatilidad se modela como una serie no observable o latente, mientras que los datos observados se modelan como una función de la volatilidad latente.

El Modelo de Espacio de Estados con Volatilidad Estocástica se define por las siguientes ecuaciones:

1. Ecuación de Observación:

Y\_t = Z\_t \* alpha\_t + epsilon\_t

Donde Y\_t es la observación en el tiempo t, Z\_t es la matriz de diseño que relaciona la volatilidad latente con las observaciones, alpha\_t es el vector de estados latentes en el tiempo t y epsilon\_t es el término de error de observación.

2. Ecuación de Transición:

alpha\_t = T\_t \* alpha\_{t-1} + R\_t \* eta\_t

Donde T\_t es la matriz de transición que describe cómo evolucionan los estados latentes en el tiempo, R\_t es la matriz de selección que describe cómo la volatilidad latente se ve afectada por el proceso estocástico y eta\_t es el término de error de transición.

3. Ecuación de la Volatilidad:

sigma\_t^2 = exp(Q\_t)

Donde sigma\_t^2 es la varianza de la volatilidad latente en el tiempo t y Q\_t es el estado latente de la volatilidad.

El objetivo principal del modelo es estimar los estados latentes y la volatilidad no observada en función de los datos observados. Esto se puede lograr utilizando técnicas de estimación como el filtro de Kalman, el algoritmo de suavizado y el algoritmo de máxima verosimilitud.

Al estimar el Modelo de Espacio de Estados con Volatilidad Estocástica, se obtiene una estimación de la volatilidad latente a lo largo del tiempo, lo que permite realizar pronósticos más precisos y capturar mejor la naturaleza cambiante de la volatilidad en el criptotrading.

Es importante tener en cuenta que la implementación detallada del modelo puede variar según la biblioteca o el paquete que se utilice. Existen varias implementaciones en Python, como `statsmodels` y `pykalman`, que proporcionan funciones y clases específicas para estimar y analizar modelos de espacio de estados con volatilidad estocástica.

Un código sencillo, con la librería pykalman, sería el siguiente:

import numpy as np

import matplotlib.pyplot as plt

from pykalman import KalmanFilter

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados con volatilidad estocástica

transition\_matrices = [[0.8]]

transition\_covariance = [[0.1]]

observation\_matrices = [[1]]

observation\_covariance = [[1]]

initial\_state\_mean = [0]

initial\_state\_covariance = [[1]]

n\_dim\_state = 1

kf = KalmanFilter(transition\_matrices=transition\_matrices,

observation\_matrices=observation\_matrices,

transition\_covariance=transition\_covariance,

observation\_covariance=observation\_covariance,

initial\_state\_mean=initial\_state\_mean,

initial\_state\_covariance=initial\_state\_covariance,

n\_dim\_state=n\_dim\_state)

# Ajustar el modelo a los datos

filtered\_state\_means, filtered\_state\_covariances = kf.filter(y)

# Obtener la volatilidad estimada

volatility = np.sqrt(filtered\_state\_covariances[:, 0, 0])

# Plotear los datos y la volatilidad estimada

plt.figure(figsize=(10, 6))

plt.plot(t, y, label='Datos')

plt.plot(t, volatility, label='Volatilidad Estimada')

plt.legend()

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.title('Modelo de Espacio de Estados con Volatilidad Estocástica')

plt.show()

En este ejemplo, se generan datos sintéticos utilizando un proceso AR(1). Luego, se define el modelo de espacio de estados con volatilidad estocástica utilizando los parámetros de transición y covarianza, matrices de observación y los valores iniciales. Se utiliza la clase KalmanFilter de la biblioteca pykalman para construir el modelo. A continuación, se ajusta el modelo a los datos utilizando el método filter(), lo cual proporciona los estados filtrados (mean) y las covarianzas correspondientes. Se obtiene la volatilidad estimada extrayendo la desviación estándar de la covarianza del estado latente. Finalmente, se plotean los datos originales y la volatilidad estimada a lo largo del tiempo.

Ahora veamos un código con la librería statsmodels:

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Generar datos sintéticos

np.random.seed(0)

n = 200

t = np.arange(n)

y = np.random.normal(loc=0, scale=1, size=n)

for i in range(1, n):

y[i] = 0.8 \* y[i-1] + np.random.normal()

# Definir el modelo de espacio de estados con volatilidad estocástica

model = sm.tsa.UnobservedComponents(y, 'rwalk', stochastic\_level=True)

# Ajustar el modelo a los datos

result = model.fit()

# Obtener la volatilidad estimada

volatility = result.level.apply(np.sqrt)

# Plotear los datos y la volatilidad estimada

plt.figure(figsize=(10, 6))

plt.plot(t, y, label='Datos')

plt.plot(t, volatility, label='Volatilidad Estimada')

plt.legend()

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.title('Modelo de Espacio de Estados con Volatilidad Estocástica')

plt.show()

En este ejemplo, también se generan datos sintéticos utilizando un proceso AR(1). Luego, se utiliza la clase UnobservedComponents de statsmodels.tsa para definir el modelo de espacio de estados con volatilidad estocástica. El argumento 'rwalk' especifica que la tendencia es un paseo aleatorio y se establece stochastic\_level=True para modelar la volatilidad como un componente estocástico. A continuación, se ajusta el modelo a los datos utilizando el método fit(). Se obtiene la volatilidad estimada a través del atributo level del resultado del ajuste y se calcula la raíz cuadrada para obtener la desviación estándar de la volatilidad. Finalmente, se plotean los datos originales y la volatilidad estimada a lo largo del tiempo.

Este método parece que no realiza predicciones de precios, sino más bien analiza la volatilidad del mercado. Habrá que ver si en realidad nos funciona el método.

Los códgos no se han probado.

###### 1.1.2.4.4.- Modelo de espacio de estados no lineal:

El Modelo de Espacio de Estados No Lineal (Nonlinear State Space Model en inglés) es una técnica de modelado que permite capturar relaciones no lineales entre las variables observadas y las variables latentes en un sistema. A diferencia de los modelos lineales de espacio de estados, que asumen relaciones lineales entre las variables, los modelos no lineales permiten representar de manera más flexible las interacciones complejas y no lineales presentes en muchos fenómenos.

En un Modelo de Espacio de Estados No Lineal, las variables observadas y las variables latentes se relacionan mediante funciones no lineales. Estas funciones no lineales pueden ser cualquier función matemática que refleje la relación entre las variables. Por ejemplo, se pueden utilizar funciones polinómicas, exponenciales, trigonométricas u otras funciones no lineales para modelar las relaciones.

La formulación general de un Modelo de Espacio de Estados No Lineal se puede expresar como:

- Equación de estado:

x(t) = f(x(t-1), u(t), θ) + ε(t)

- Equación de observación:

y(t) = g(x(t), θ) + δ(t)

Donde:

- x(t) es el vector de variables latentes en el tiempo t.

- f() es una función no lineal que describe la evolución de las variables latentes en el tiempo.

- u(t) es el vector de entradas externas en el tiempo t.

- θ es el vector de parámetros del modelo.

- ε(t) es el término de error en la ecuación de estado.

- y(t) es el vector de variables observadas en el tiempo t.

- g() es una función no lineal que relaciona las variables latentes con las variables observadas.

- δ(t) es el término de error en la ecuación de observación.

El objetivo en el Modelado de Espacio de Estados No Lineal es estimar los valores de las variables latentes x(t) y los parámetros θ a partir de las observaciones y. Esto se logra mediante técnicas de estimación y ajuste de los parámetros, como el método de máxima verosimilitud o el filtro de Kalman extendido.

Una vez que se ha estimado el Modelo de Espacio de Estados No Lineal, se pueden utilizar para predecir valores futuros de las variables observadas y analizar el comportamiento del sistema en función de los valores de las variables latentes.

Es importante tener en cuenta que la estimación y el ajuste de los Modelos de Espacio de Estados No Lineal pueden ser computacionalmente intensivos debido a la naturaleza no lineal de las funciones involucradas. Por lo tanto, es común utilizar métodos numéricos y algoritmos de optimización para encontrar los mejores valores de los parámetros y realizar las predicciones.

En resumen, el Modelo de Espacio de Estados No Lineal es una poderosa herramienta de modelado que permite capturar relaciones no lineales en sistemas dinámicos. Su flexibilidad y capacidad para modelar fenómenos complejos lo hacen adecuado para una amplia gama de aplicaciones, incluido el análisis y la predicción en el campo del criptotrading.

Vemos que este método puede ser costoso computacionalmente, sin embargo, también vemos que es una buena opción de predicción en sistemas no lineales, siendo nuestros datos de naturaleza no lineal. Sería buena opción a la hora de la predicción.

Un códifo básico, con la librería pykalman, sería como sigue:

import numpy as np

from pykalman import UnscentedKalmanFilter

# Definir la función no lineal para la evolución del estado

def state\_transition\_function(state, noise):

return state + 0.1 \* state\*\*2 + noise

# Definir la función no lineal para la observación del estado

def observation\_function(state, noise):

return state\*\*3 + noise

# Generar datos sintéticos

np.random.seed(0)

n = 100

state = np.zeros(n)

observations = np.zeros(n)

for i in range(1, n):

state[i] = state\_transition\_function(state[i-1], np.random.normal(0, 0.1))

observations[i] = observation\_function(state[i], np.random.normal(0, 1))

# Definir el modelo de espacio de estados no lineal

kf = UnscentedKalmanFilter(

state\_transition\_func=state\_transition\_function,

observation\_func=observation\_function

)

# Ajustar el modelo a los datos

filtered\_state\_estimates, \_ = kf.filter(observations)

# Imprimir los resultados

print("Estimaciones de estado filtradas:")

print(filtered\_state\_estimates)

En este ejemplo, se define una función no lineal para la evolución del estado (state\_transition\_function) y una función no lineal para la observación del estado (observation\_function). Estas funciones representan las relaciones no lineales entre las variables latentes y las variables observadas en el modelo.

Luego, se generan datos sintéticos utilizando estas funciones y ruido gaussiano. A continuación, se crea una instancia de la clase UnscentedKalmanFilter de pykalman, pasando las funciones de evolución del estado y observación del estado como argumentos.

Finalmente, se ajusta el modelo a los datos utilizando el método filter y se obtienen las estimaciones filtradas de los estados latentes.

Recuerda que este es solo un ejemplo básico y que la implementación del Modelo de Espacio de Estados No Lineal puede variar según tus necesidades y los datos específicos que estés utilizando. Asegúrate de consultar la documentación de pykalman para obtener más información sobre los parámetros y opciones disponibles.

El código no se ha probado.

###### 1.1.2.4.5.- Modelo de Espacio de Estados con Cambio de Nivel:

Este es el último ejemplo de los “state space models”

El Modelo de Espacio de Estados con Cambio de Nivel es una técnica utilizada para modelar series de tiempo que presentan cambios abruptos o cambios estructurales en su nivel a lo largo del tiempo. Este modelo permite capturar y estimar de manera flexible estos cambios, lo que lo hace útil en el análisis de datos de series temporales que exhiben comportamientos no estacionarios.

El modelo se compone de dos componentes principales: el componente de evolución del estado y el componente de observación. El componente de evolución del estado describe cómo evoluciona el estado latente a lo largo del tiempo, mientras que el componente de observación relaciona el estado latente con las observaciones disponibles.

El componente de evolución del estado se define mediante la siguiente ecuación:

S\_t = S\_{t-1} + \beta D\_t + \epsilon\_t

donde S\_t es el estado latente en el tiempo t, S\_{t-1} es el estado latente en el tiempo t-1, \beta es el cambio de nivel, D\_t es una variable binaria que indica si hay un cambio de nivel en el tiempo t, y \epsilon\_t es el ruido de evolución del estado.

El componente de observación se define mediante la siguiente ecuación:

Y\_t = S\_t + \eta\_t

donde Y\_t es la observación en el tiempo t y \eta\_t es el ruido de observación.

El modelo se estima utilizando métodos de estimación de máxima verosimilitud o métodos bayesianos. Los parámetros estimados incluyen el cambio de nivel (\beta), los estados latentes iniciales, los ruidos de evolución y de observación, y la distribución inicial de los estados latentes.

Una vez que el modelo ha sido ajustado a los datos, se pueden obtener las estimaciones de los estados latentes, lo que proporciona información sobre la tendencia y los cambios estructurales en la serie de tiempo. También se pueden realizar predicciones futuras utilizando el modelo ajustado.

En resumen, el Modelo de Espacio de Estados con Cambio de Nivel es una herramienta útil para modelar series de tiempo con cambios abruptos en su nivel. Permite capturar y estimar estos cambios, lo que proporciona información valiosa sobre la dinámica y las tendencias en los datos de series temporales.

Un código básico sería el siguiente:

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Generar datos sintéticos con cambio de nivel

np.random.seed(0)

n = 200

time = np.arange(n)

level\_change\_point = 100

level1 = 10

level2 = 15

noise = np.random.normal(0, 1, n)

observations = np.concatenate([np.repeat(level1, level\_change\_point), np.repeat(level2, n - level\_change\_point)]) + noise

# Definir el modelo de espacio de estados con cambio de nivel

model = sm.tsa.UnobservedComponents(observations, 'llevel')

# Ajustar el modelo a los datos

result = model.fit()

# Obtener las estimaciones de los estados latentes

filtered\_state = result.filtered\_state

# Plotear los datos y las estimaciones de nivel

plt.plot(time, observations, label='Observaciones')

plt.plot(time, filtered\_state.filtered\_state[0], label='Estimaciones de nivel')

plt.axvline(x=level\_change\_point, color='red', linestyle='--', label='Punto de cambio de nivel')

plt.xlabel('Tiempo')

plt.ylabel('Valor')

plt.legend()

plt.show()

En este ejemplo, se generan datos sintéticos que presentan un cambio de nivel en el punto de tiempo 100. Luego, se define el modelo de espacio de estados con cambio de nivel utilizando la clase UnobservedComponents de statsmodels. El argumento 'llevel' indica que se está modelando un cambio de nivel en el nivel de la serie de tiempo.

A continuación, el modelo se ajusta a los datos utilizando el método fit(). Las estimaciones de los estados latentes se obtienen a través del atributo filtered\_state.filtered\_state.

Finalmente, se traza el gráfico de las observaciones originales y las estimaciones del nivel, destacando el punto de cambio de nivel con una línea roja discontinua.

##### 1.1.2.5.- GARCH

El modelo GARCH (Generalized Autoregressive Conditional Heteroskedasticity) es una técnica utilizada en el análisis de series temporales para modelar y predecir la volatilidad de los datos. Fue propuesto por Robert Engle en 1982 como una extensión del modelo ARCH (Autoregressive Conditional Heteroskedasticity).

El modelo GARCH es especialmente útil cuando los datos presentan volatilidad heteroscedástica, es decir, cuando la varianza de los errores no es constante a lo largo del tiempo. Este modelo permite capturar y modelar esta variación en la volatilidad a través del tiempo.

La ecuación general del modelo GARCH(p, q) se puede expresar de la siguiente manera:

σ²\_t = ω + ∑(α\_i \* ε²\_{t-i}) + ∑(β\_j \* σ²\_{t-j})

Donde:

- σ²\_t es la varianza condicional en el tiempo t.

- ω es el parámetro de constante.

- α\_i son los parámetros de autorregresión de los errores al cuadrado.

- ε²\_{t-i} son los errores al cuadrado en el tiempo t-i.

- β\_j son los parámetros de autorregresión de las varianzas condicionales.

- σ²\_{t-j} son las varianzas condicionales en el tiempo t-j.

El modelo GARCH(p, q) se caracteriza por tener un componente autoregresivo (AR) para los errores al cuadrado y otro componente autoregresivo (AR) para las varianzas condicionales. Los parámetros α\_i y β\_j controlan la contribución de los errores pasados y las varianzas pasadas, respectivamente, en la volatilidad actual. El parámetro ω representa la varianza constante.

Para estimar los parámetros del modelo GARCH, se utiliza generalmente el método de máxima verosimilitud. Este método busca encontrar los valores de los parámetros que maximizan la probabilidad de obtener los datos observados dados los parámetros del modelo.

Una vez que el modelo GARCH ha sido estimado, se pueden obtener las estimaciones de la volatilidad condicional para cada período de tiempo. Estas estimaciones proporcionan información valiosa sobre la incertidumbre y la variabilidad en los datos. Además, se pueden realizar pronósticos de la volatilidad futura utilizando el modelo ajustado.

El modelo GARCH ha demostrado ser útil en una variedad de aplicaciones, incluyendo el análisis de mercados financieros, donde la volatilidad es un factor importante a tener en cuenta en la toma de decisiones de inversión. Al modelar y predecir la volatilidad, el modelo GARCH puede ayudar a los inversores a evaluar el riesgo y ajustar sus estrategias de inversión en consecuencia.

Como podemos ver, este modelo solo presenta un análisis sobre la volatilidad del mercado, lo cual puede ayudar a la gestión de riesgos, pero no parece ser un buen candidato para la predicción de precios, sino solo en la predicción de la volatilidad futura.

Un código sencillo de esto sería:

import pandas as pd

from arch import arch\_model

# Cargar los datos de volatilidad (por ejemplo, precios históricos)

data = pd.read\_csv('ruta\_del\_archivo.csv', index\_col='Fecha', parse\_dates=True)

# Crear el modelo GARCH(1, 1)

model = arch\_model(data['Volatilidad'], vol='Garch', p=1, q=1)

# Ajustar el modelo a los datos

model\_fit = model.fit()

# Obtener las predicciones de volatilidad condicional

predictions = model\_fit.conditional\_volatility

# Imprimir las predicciones de volatilidad

print(predictions)

El modelo GARCH se crea utilizando arch\_model de la librería arch. En este ejemplo, se especifica vol='Garch' para indicar que queremos un modelo GARCH. Los parámetros p=1 y q=1 indican que estamos utilizando un modelo GARCH(1, 1).

Después de ajustar el modelo a los datos utilizando fit(), podemos acceder a las predicciones de volatilidad condicional utilizando conditional\_volatility. Estas predicciones representan la volatilidad estimada para cada observación en los datos.

##### 1.1.2.6.- Prophet

Prophet no es un método de ML, sino una librería creada por facebook para el análisis de series temporales. Como tal, vamos a dar una descripción breve a continuación sobre esta librería. Posteriormente voy a dejar una ruta de trabajo recomendada para la librería.

Prophet fue diseñada específicamente para ser fácil de usar y brindar resultados rápidos y precisos en la predicción de datos de series de tiempo. Aunque Prophet no fue creado específicamente para el criptotrading, puede ser utilizado para hacer pronósticos de precios de criptomonedas.

A continuación, se describen algunos aspectos clave de Prophet y cómo se puede utilizar para el criptotrading:

Modelo aditivo: Prophet asume que la serie de tiempo es una combinación aditiva de diferentes componentes, incluyendo la tendencia, las estacionalidades y los efectos de vacaciones. Esta característica permite capturar diferentes patrones presentes en los datos de criptomonedas.

Componentes principales: Prophet descompone la serie de tiempo en sus componentes principales, es decir, la tendencia y las estacionalidades. Estos componentes se modelan por separado y luego se combinan para generar las predicciones.

Detección automática de cambios: Prophet tiene la capacidad de detectar automáticamente cambios en la tendencia y los efectos de estacionalidad, lo que es especialmente útil en el criptotrading, donde los precios pueden experimentar cambios bruscos y repentinos.

Flexibilidad en los datos de entrada: Prophet acepta como entrada un DataFrame de Pandas con dos columnas obligatorias: 'ds' (las fechas) y 'y' (los valores de la serie de tiempo). Esto facilita el proceso de preparación de los datos antes de aplicar el modelo.

Personalización de los modelos: Prophet proporciona la posibilidad de personalizar los modelos ajustando diferentes hiperparámetros, como la estacionalidad, la suavidad de la tendencia y la sensibilidad a los cambios. Esto permite adaptar el modelo a las características específicas de los datos de criptomonedas.

Visualización de resultados: Prophet ofrece herramientas para visualizar los resultados de las predicciones, incluyendo gráficos interactivos con los datos históricos y las predicciones futuras. Esto ayuda a los traders a comprender y evaluar los resultados de manera intuitiva.

En resumen, Prophet es una herramienta útil para el criptotrading que simplifica el proceso de análisis y predicción de series de tiempo. Puede ayudar a los traders a identificar patrones, tendencias y cambios en los precios de las criptomonedas, brindando información valiosa para la toma de decisiones de inversión. Sin embargo, es importante utilizar Prophet en conjunto con otras técnicas y análisis para obtener una visión más completa y precisa del mercado de criptomonedas.

Ahora veamos la ruta de trabajo...

1. Recopilación de datos: Obtén los datos históricos de precios de la criptomoneda que deseas analizar. Puedes obtener estos datos de diversas fuentes, como exchanges de criptomonedas o servicios de datos financieros.

2. Preparación de datos: Prepara los datos en un formato adecuado para Prophet. Debes tener una columna con las fechas ('ds') y otra columna con los precios ('y'). Asegúrate de que los datos estén en formato de serie de tiempo y que no haya valores faltantes.

3. División de datos: Divide los datos en conjuntos de entrenamiento y prueba. Puedes utilizar una proporción como 80% para entrenamiento y 20% para prueba. Esto te permitirá evaluar el rendimiento del modelo en datos no vistos.

4. Creación del modelo: Crea un objeto de modelo Prophet utilizando la clase `Prophet` de la librería. Puedes ajustar diversos parámetros para personalizar el modelo, como la configuración de la estacionalidad, la inclusión de días festivos, entre otros.

5. Ajuste del modelo: Utiliza el método `fit` para ajustar el modelo a los datos de entrenamiento. Esto calculará los parámetros del modelo basados en los datos históricos.

6. Generación de predicciones: Utiliza el método `predict` para generar predicciones futuras del precio de la criptomoneda. Puedes especificar el número de períodos hacia adelante que deseas pronosticar.

7. Evaluación del modelo: Evalúa el rendimiento del modelo utilizando los datos de prueba. Compara las predicciones generadas por el modelo con los valores reales y calcula métricas de evaluación, como el error cuadrado medio (RMSE) o el error absoluto medio (MAE).

8. Visualización de resultados: Utiliza las herramientas de visualización de Prophet para representar gráficamente los datos históricos, las predicciones y los intervalos de confianza. Esto te ayudará a comprender la tendencia y la incertidumbre en las predicciones.

9. Implementación de estrategias de trading: Utiliza las predicciones generadas por el modelo para implementar estrategias de trading. Por ejemplo, puedes generar señales de compra o venta basadas en la dirección de las predicciones y establecer reglas específicas para tomar decisiones comerciales.

Es recomendable utilizar Prophet junto con otras técnicas de análisis y considerar factores adicionales, como el análisis fundamental y las condiciones del mercado, al tomar decisiones de trading.

A mi parecer, esta librería podría llegar a ser muy rígida con respecto a lo que necesitemos, sin embargo, también podría tener herramientas útiles que pudiéramos utilizar en un futuro, por lo que sería buena idea preguntarnos si vale la pena investigarla y decidir si dejarla de lado o no.

#### Análisis de Hiperparámetros:

Un hiperparámetro es un parámetro externo a un modelo de aprendizaje automático que se elige antes de iniciar el proceso de entrenamiento. A diferencia de los parámetros del modelo, que se aprenden durante el entrenamiento, los hiperparámetros son configurables y se establecen antes de que comience el proceso de optimización.

Los hiperparámetros definen aspectos del algoritmo de aprendizaje automático y afectan el rendimiento y el comportamiento del modelo. Son decisiones que toma el desarrollador o el científico de datos para configurar el algoritmo y personalizarlo para un problema y conjunto de datos específicos.

Algunos ejemplos comunes de hiperparámetros incluyen:

1. Tasa de aprendizaje: Es un hiperparámetro que determina la cantidad de ajuste que se realiza durante cada iteración del algoritmo de optimización. Controla la velocidad a la que el modelo aprende de los datos.

2. Número de capas ocultas: En las redes neuronales, el número de capas ocultas es un hiperparámetro que define la cantidad de capas intermedias entre la capa de entrada y la capa de salida. Este hiperparámetro afecta la complejidad y capacidad del modelo para aprender representaciones más abstractas.

3. Número de árboles en un bosque aleatorio: En algoritmos de ensamble como Random Forest, el número de árboles es un hiperparámetro que determina la cantidad de árboles de decisión utilizados en el modelo.

4. Tamaño del lote (batch size): En el entrenamiento de redes neuronales, el tamaño del lote es el número de muestras de entrenamiento que se utilizan en cada iteración del algoritmo de optimización. Es un hiperparámetro que puede afectar la estabilidad y velocidad del entrenamiento.

5. Regularización: La regularización es una técnica utilizada para evitar el sobreajuste en los modelos de aprendizaje automático. Los hiperparámetros de regularización, como la fuerza de la regularización L1 o L2, controlan la cantidad de penalización aplicada a los coeficientes del modelo.

La selección adecuada de los hiperparámetros es esencial para obtener un modelo de aprendizaje automático bien ajustado y con un buen rendimiento. Por lo general, se realizan experimentos y validaciones cruzadas con diferentes combinaciones de valores de hiperparámetros para encontrar la configuración óptima. Existen técnicas de optimización de hiperparámetros, como la búsqueda de cuadrícula (grid search) y la optimización bayesiana, que ayudan a encontrar los mejores valores de hiperparámetros de manera más eficiente..