Bu modelin, veriler gerçekleşmeden önce bir tahmin yapabildiğini ispatlamak için aşağıdaki hususlara dikkat edilmesi önemlidir:

1. **Veri Seti Bölünmesi (Train-Test Split):** Kodda, veriler "eğitim" ve "test" olarak ikiye ayrılıyor. Eğitim verisi modelin eğitilmesi için kullanılırken, test verisi modelin doğruluğunu değerlendirmek için ayrılmıştır. Test seti, modelin hiç görmediği verilerden oluşur ve bu nedenle geçmişteki verilerle eğitilen model, gelecekteki verileri tahmin etmeye çalışır. Bu durum, modelin geçmişi kullanarak geleceği tahmin etmeye çalıştığını gösterir.

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

Bu kod satırı veri setini %80 eğitim, %20 test olacak şekilde bölüyor. Test verisi, modelin eğitim sırasında görmediği verilerdir, dolayısıyla test verisi üzerinde yapılan tahmin, gerçekleşmeden önce yapılan bir tahmin niteliğindedir.

1. **Gün Kaydırma (Shift):** df['preds'] = df['Close'].shift(-num) satırı, bir gün sonrasının kapanış fiyatını tahmin etmek için kullanılır. Bu da modelin gelecekteki kapanış fiyatlarını öngörmek üzere eğitildiğini gösterir. Burada num kadar gün ilerisine kaydırarak, gelecekteki bir değeri tahmin etmeyi amaçlıyoruz.
2. **Test Verisi Üzerinde Tahmin Yapılması:** Test verisi üzerinde yapılan tahminler, modelin eğitildiği veriler dışında, o gün gerçekleşmeden önceki kapanış fiyatlarının tahmin edilmesini sağlar. Test setindeki X\_test üzerinde yapılan tahmin şu şekilde yapılır:

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

Bu satır, gelecekteki kapanış fiyatını tahmin etmek için eğitilmiş modelle yapılır ve test verisindeki değerlere dayanmaktadır. Bu nedenle bu tahminler, modelin gelecekte gerçekleşecek olayları önceden tahmin etme kapasitesini test eder.

1. **Son Günü ve Tahmini Karşılaştırma:** Kodda ayrıca son günün tahmin edilen ve gerçekleşen fiyatı karşılaştırılarak tahminlerin doğruluğu gösterilmektedir:

son\_gun\_gercek = results\_df.iloc[-1]['Gerçek Değerler']

son\_gun\_tahmin = results\_df.iloc[-1]['Tahmin Edilen Değerler']

print(f"Son gün gerçekleşen fiyat: {son\_gun\_gercek}")

print(f"Son gün tahmin edilen fiyat: {son\_gun\_tahmin}")

Bu karşılaştırma, modelin geleceği tahmin etmek için ne kadar başarılı olduğunu gözlemlememizi sağlar.

**Özetle:** Bu kod, geçmiş verilerle eğitilen modelin, gelecekte gerçekleşecek olan kapanış fiyatını tahmin etmesini sağlıyor. Test seti, modelin eğitimi sırasında kullanılmayan verilerden oluşuyor, bu da tahminlerin tamamen geçmiş bilgilere dayanarak yapıldığını ve test setindeki günler henüz gerçekleşmeden önce bir tahmin yaptığını gösteriyor.

**Şunu dedin:**

Test seti, modelin eğitimi sırasında kullanılmayan verilerden oluşuyor, dedin. eğitimde kullanılmayan verileri nereden buluyor

**Cevap:** Eğitim ve test verilerinin ayrılma sürecinde modelin eğitimi sırasında kullanılmayan veriler, veri setinin bölünmesiyle elde edilir. Bunu sağlamak için train\_test\_split işlevi kullanılır.

Şu satırda:

python

Kodu kopyala

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

Bu satır, veri setini %80 eğitim ve %20 test olarak böler. X\_train ve y\_train modelin öğrenmesi için kullanılırken, X\_test ve y\_test test verileridir. Bu test verileri modelin eğitimi sırasında kullanılmaz, dolayısıyla model bu verileri daha önce hiç görmemiştir.

Eğitim sürecinde model X\_train ve y\_train verilerini kullanarak kendini ayarlar. Daha sonra X\_test verileri üzerinde tahminler yapılır ve bu tahminler y\_test ile karşılaştırılarak modelin doğruluğu hesaplanır.

Bu test verileri modelin geleceği önceden görmeden tahmin yapma yeteneğini değerlendirmek için kullanılır. Test setinin eğitim sırasında kullanılmaması, modelin daha önce hiç görmediği yeni verilerde ne kadar başarılı olduğunu gösterir ve bu, gelecekteki fiyatları tahmin etme kapasitesinin bir göstergesi olarak kullanılır.

Kod şu şekildeydi:

import pandas as pd

import yfinance as yf

from sklearn.preprocessing import StandardScaler

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

from sklearn.linear\_model import LinearRegression

import numpy as np

# Örnek olarak bir veri seti yükleyelim

ticker = "BTC-USD"  # Bitcoin veri kümesini örnek alıyoruz

data = yf.download(ticker, start="2024-09-01", end="2024-11-05")

df = pd.DataFrame(data)

# Bu satırlar, yfinance kütüphanesi aracılığıyla belirli bir tarih aralığındaki Bitcoin fiyat verilerini Yahoo Finance'ten indirir ve bir DataFrame’e (df) aktarır.

# Veri, günlük açılış (Open), kapanış (Close), en yüksek (High), en düşük (Low) ve hacim (Volume) gibi sütunları içerir.

# Hedef değişkeni ekleyin, uyarıyı önlemek için .loc kullanıyoruz

num = 1  # Tahmin yapılacak gün sayısı

df.loc[:, 'preds'] = df['Close'].shift(-num)

# Bu satır, kapanış fiyatlarını num kadar kaydırarak (örneğin num = 1 için bir gün) hedef değişken (preds) oluşturur. Bu, bir gün sonrasının kapanış fiyatını tahmin etmeyi sağlar.

# Kaydırma (shift) işlemi sonrasında oluşan NaN değerleri kaldırmak için df.dropna(inplace=True) kullanılır.

# NaN değerleri kaldırın (shift işlemi sonucu oluşur)

df.dropna(inplace=True)

# Özellik ve hedef ayrımı

X = df[['Open', 'High', 'Low', 'Volume']]

y = df['preds']

# Özellikler (X): Modelin tahmin yaparken kullanacağı giriş verileridir. Burada açılış, en yüksek, en düşük fiyatlar ve hacim (Volume) özellik olarak seçilmiştir.

# Hedef (y): Modelin tahmin etmeye çalıştığı değerdir, burada preds yani bir gün sonrası kapanış fiyatı.

# Veriyi ölçekleyin

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Bu adımda, özelliklerin (X) daha iyi performans için standartlaştırılması (ölçeklenmesi) sağlanır.

# StandardScaler, her bir özelliği ortalaması 0, standart sapması 1 olacak şekilde dönüştürür.

# Eğitim ve test setlerini ayırın

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

# Veri seti %80 eğitim ve %20 test olacak şekilde bölünür.

# Eğitim verisi modelin öğrenmesi için kullanılırken, test verisi modelin doğruluğunu değerlendirmek için ayrılır.

# Modeli oluşturun ve eğitin

model = LinearRegression()

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

# Linear Regression (Doğrusal Regresyon) modeli oluşturulur ve fit metodu ile eğitim verileri üzerinde model eğitilir.

# Model, fiyatlar arasında doğrusal bir ilişki olduğunu varsayar.

# Model doğruluğunu kontrol edin

accuracy = model.score(X\_test, y\_test)

print(f"Model doğruluğu: {accuracy}")

# model.score fonksiyonu, modelin doğruluğunu (burada R^2 skoru) test verileri üzerinde hesaplar.

# Örneğin, modelin veriyi %98.75 oranında açıkladığı anlamına gelebilir. Yani, model verilerdeki varyansın büyük bir kısmını yakalamaktadır.

# Tahmin yapın (örneğin, test seti üzerinden tahmin yapıyoruz)

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

# Tahmin ve gerçek değerleri bir DataFrame'e ekleyin

results\_df = pd.DataFrame({'Gerçek Değerler': y\_test, 'Tahmin Edilen Değerler': y\_pred})

results\_df.reset\_index(drop=True, inplace=True)

# Test seti üzerindeki gerçek ve tahmin edilen kapanış fiyatlarını karşılaştırmak için bir DataFrame oluşturulur.

# Son günü ve tahminini görüntüleyin

son\_gun\_tarihi = df.index[-1]  # Son günün tarihini alıyoruz

son\_gun\_gercek = results\_df.iloc[-1]['Gerçek Değerler']

son\_gun\_tahmin = results\_df.iloc[-1]['Tahmin Edilen Değerler']

print(f"Son gün tarihi: {son\_gun\_tarihi}")

print(f"Son gün gerçekleşen fiyat: {son\_gun\_gercek}")

print(f"Son gün tahmin edilen fiyat: {son\_gun\_tahmin}")

# Son gündeki tahmin edilen kapanış fiyatı ve gerçekleşen kapanış fiyatı, ayrıca tarih ekrana yazdırılır.

# Tüm tahmin ve gerçekleşen fiyatlar

print(results\_df)

# Tüm test seti için tahmin ve gerçek kapanış fiyatlarını gösterir.

# Sonuçları gösteren DataFrame'i kaydetmek için

tablo\_adresi = 'btc\_prediction\_results.csv'

results\_df.to\_csv(tablo\_adresi, index=False)

print(f"Sonuçlar CSV olarak kaydedildi: {tablo\_adresi}")

# Tahmin ve gerçek değerleri içeren DataFrame bir CSV dosyasına kaydedilir.

Ve son gün olan 05 i vermiyordu 03’ü veriyordu ve şöyle değişiklik yaptım:

* + son\_gun\_tarihi = df.index[-1] satırını değiştirdim.
  + Bunun yerine, son\_gun\_tarihi = df.index[-(num + 1)] ifadesini kullandım. Bu değişiklik, tahmin edilen günün tarihini almak için kullanıldı. Son kapanış gününü değil, tahmin yapmak istediğiniz tarihten bir gün öncesini ifade etmek için kaydırma uygulandı.

1. **Son Gün Tarih Mesajı:**
   * print(f"Son gün tarihi: {son\_gun\_tarihi}") satırını şu şekilde değiştirdim: print(f"Tahmin edilen gün tarihi: {son\_gun\_tarihi}").
   * Bu şekilde mesajın anlamını daha net hale getirdim; tahmin edilen günün tarihi belirtildi.

Bu değişiklikler sayesinde, tahmin edilen tarih (2024-11-05) doğru bir şekilde alınır ve çıktı olarak gösterilir.

Hale geldi.

Ama hayır hayır bunda da son tarih olarak 04’ü verdi.

Yine değişiklikkler yapıyoruz kurcalıyoruz ve

 İlk önce 4 Kasım için tahmin yapılıyor

 Sonra 4 Kasım tahmini kullanılarak 5 Kasım için tahmin yapılıyor

 5 Kasım tahmini için:

* Open değeri olarak 4 Kasım tahmini kullanılıyor
* High değeri olarak tahmin değerinin %1 üstü kullanılıyor
* Low değeri olarak tahmin değerinin %1 altı kullanılıyor
* Volume değeri olarak son günün hacmi kullanılıyor

Şimdi çalışan kodu veriyorum ve bu kod 09’u falan tahmin ediyor istediğin güne kadar gidiyor tahmin ede ede.

import pandas as pd

import yfinance as yf

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

import numpy as np

from datetime import datetime, timedelta

def predict\_next\_days(end\_date, df, model, scaler):

    # Son gerçek veri tarihini al ve timezone'u kaldır

    last\_real\_date = df.index[-1].tz\_localize(None)

    # Hedef tarihe kadar kaç gün olduğunu hesapla

    target\_date = datetime.strptime(end\_date, '%Y-%m-%d')

    days\_to\_predict = (target\_date - last\_real\_date).days

    if days\_to\_predict <= 0:

        return pd.DataFrame()  # Hedef tarih geçmiş tarihse boş DataFrame dön

    predictions = []

    dates = []

    current\_features = df[['Open', 'High', 'Low', 'Volume']].iloc[-1:].values

    for i in range(days\_to\_predict):

        # Mevcut özellikleri ölçekle ve tahmin yap

        current\_scaled = scaler.transform(current\_features)

        next\_price = model.predict(current\_scaled)[0]

        # Sonuçları kaydet

        next\_date = last\_real\_date + timedelta(days=i+1)

        predictions.append(next\_price)

        dates.append(next\_date.strftime('%Y-%m-%d'))

        # Bir sonraki tahmin için özellikleri güncelle

        current\_features = np.array([

            [float(next\_price),  # Open

             float(next\_price \* 1.01),  # High

             float(next\_price \* 0.99),  # Low

             float(current\_features[0][3])]  # Volume (son bilinen hacmi kullan)

        ])

    return pd.DataFrame({

        'Tarih': dates,

        'Tahmin Edilen Fiyat': predictions

    })

# Veri setini yükle

ticker = "BTC-USD"

data = yf.download(ticker, start="2024-04-01", end="2024-11-09")

df = pd.DataFrame(data)

# Hedef değişkeni ekle

num = 1

df.loc[:, 'preds'] = df['Close'].shift(-num)

df.dropna(inplace=True)

# Özellik ve hedef ayrımı

X = df[['Open', 'High', 'Low', 'Volume']]

y = df['preds']

# Veriyi ölçekle

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Tüm veriyi eğitim için kullan

model = LinearRegression()

model.fit(X\_scaled, y)

# İstenen tarihe kadar tahmin yap

end\_date = "2024-11-09"  # İstediğiniz tarihi buraya yazabilirsiniz

predictions\_df = predict\_next\_days(end\_date, df, model, scaler)

# Model doğruluğunu hesapla

accuracy = model.score(X\_scaled, y)

# Sonuçları yazdır

print(f"Son gerçek veri tarihi: {df.index[-1].strftime('%Y-%m-%d')}")

print(f"\nTahminler:")

for \_, row in predictions\_df.iterrows():

    print(f"Tarih: {row['Tarih']}, Tahmin: ${row['Tahmin Edilen Fiyat']:.2f}")

print(f"\nModel doğruluğu: {accuracy:.4f}")

# CSV olarak kaydet

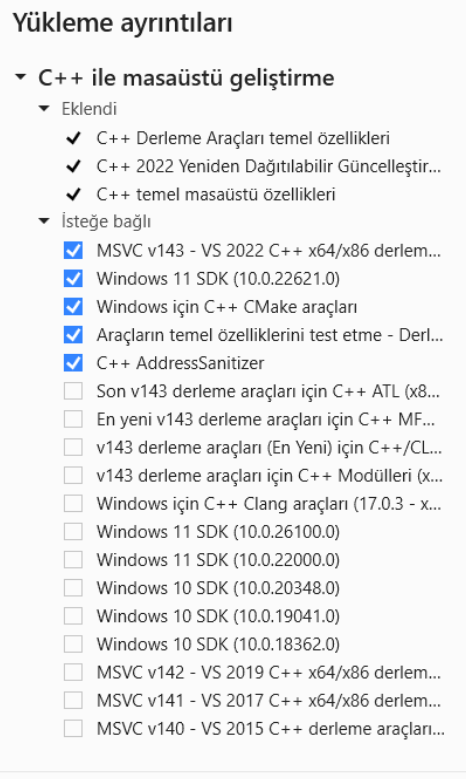
predictions\_df.to\_csv('btc\_future\_predictions.csv', index=False)

print(f"\nTahmin sonuçları 'btc\_future\_predictions.csv' dosyasına kaydedildi.")

…..

pip install PySide2

pip install PyQt5



Bunları yükledik

metin, ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tahmin uygulaması.

Kodu da bu:

import sys

import pandas as pd

import yfinance as yf

import numpy as np

from datetime import datetime, timedelta

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,

                           QPushButton, QLabel, QLineEdit, QMessageBox)

from PyQt5.QtCore import Qt

class BitcoinPredictorApp(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Bitcoin Tahmin Uygulaması")

        self.setGeometry(100, 100, 800, 600)

        # Ana widget ve layout

        main\_widget = QWidget()

        self.setCentralWidget(main\_widget)

        layout = QVBoxLayout(main\_widget)

        # Grafik için figure oluştur

        self.figure, self.ax = plt.subplots(figsize=(8, 6))

        self.canvas = FigureCanvas(self.figure)

        layout.addWidget(self.canvas)

        # Tarih giriş alanı

        date\_layout = QVBoxLayout()

        self.date\_label = QLabel("Tahmin Tarihi (YYYY-MM-DD):")

        self.date\_input = QLineEdit()

        self.date\_input.setPlaceholderText("Örnek: 2024-12-31")

        date\_layout.addWidget(self.date\_label)

        date\_layout.addWidget(self.date\_input)

        layout.addLayout(date\_layout)

        # Tahmin butonu

        self.predict\_button = QPushButton("Tahmin Et")

        self.predict\_button.clicked.connect(self.make\_prediction)

        layout.addWidget(self.predict\_button)

        # Veri ve model hazırlığı

        self.prepare\_data\_and\_model()

        self.plot\_bitcoin\_data()

    def prepare\_data\_and\_model(self):

        """Veri setini hazırla ve modeli eğit"""

        # Bitcoin verilerini çek

        self.ticker = "BTC-USD"

        start\_date = "2024-04-01"

        end\_date = datetime.now().strftime("%Y-%m-%d")

        try:

            self.df = yf.download(self.ticker, start=start\_date, end=end\_date)

            # Hedef değişkeni oluştur

            self.df.loc[:, 'preds'] = self.df['Close'].shift(-1)

            self.df.dropna(inplace=True)

            # Model için verileri hazırla

            X = self.df[['Open', 'High', 'Low', 'Volume']]

            y = self.df['preds']

            # Veriyi ölçekle

            self.scaler = StandardScaler()

            X\_scaled = self.scaler.fit\_transform(X)

            # Modeli eğit

            self.model = LinearRegression()

            self.model.fit(X\_scaled, y)

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Veri hazırlama hatası: {str(e)}")

    def plot\_bitcoin\_data(self):

        """Bitcoin verilerini grafiğe çiz"""

        try:

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax)

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği')

            self.ax.set\_xlabel('Tarih')

            self.ax.set\_ylabel('Fiyat (USD)')

            self.ax.grid(True)

            self.canvas.draw()

        except Exception as e:

            QMessageBox.warning(self, "Uyarı", f"Grafik çizim hatası: {str(e)}")

    def predict\_next\_days(self, end\_date):

        """Gelecek günler için tahmin yap"""

        last\_real\_date = self.df.index[-1].tz\_localize(None)

        target\_date = datetime.strptime(end\_date, '%Y-%m-%d')

        days\_to\_predict = (target\_date - last\_real\_date).days

        if days\_to\_predict <= 0:

            return pd.DataFrame()

        predictions = []

        dates = []

        current\_features = self.df[['Open', 'High', 'Low', 'Volume']].iloc[-1:].values

        for i in range(days\_to\_predict):

            current\_scaled = self.scaler.transform(current\_features)

            next\_price = self.model.predict(current\_scaled)[0]

            next\_date = last\_real\_date + timedelta(days=i+1)

            predictions.append(next\_price)

            dates.append(next\_date)

            current\_features = np.array([[

                float(next\_price),

                float(next\_price \* 1.01),

                float(next\_price \* 0.99),

                float(current\_features[0][3])

            ]])

        return pd.DataFrame({

            'Tarih': dates,

            'Tahmin': predictions

        })

    def make\_prediction(self):

        """Tahmin butonuna tıklandığında çalışacak fonksiyon"""

        try:

            end\_date = self.date\_input.text()

            datetime.strptime(end\_date, '%Y-%m-%d')  # Tarih formatını kontrol et

            predictions\_df = self.predict\_next\_days(end\_date)

            if predictions\_df.empty:

                QMessageBox.warning(self, "Uyarı", "Geçerli bir tarih giriniz!")

                return

            # Tahminleri grafiğe ekle

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax, label='Gerçek Veriler')

            predictions\_df.set\_index('Tarih')['Tahmin'].plot(ax=self.ax, label='Tahminler', style='--')

            self.ax.legend()

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği ve Tahminler')

            self.ax.grid(True)

            self.canvas.draw()

            # Tahmin sonuçlarını göster

            result\_text = "Tahmin Sonuçları:\n\n"

            for \_, row in predictions\_df.iterrows():

                result\_text += f"Tarih: {row['Tarih'].strftime('%Y-%m-%d')}\n"

                result\_text += f"Tahmin: ${row['Tahmin']:.2f}\n\n"

            QMessageBox.information(self, "Tahmin Sonuçları", result\_text)

        except ValueError:

            QMessageBox.warning(self, "Uyarı", "Geçersiz tarih formatı! YYYY-MM-DD formatında giriniz.")

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Tahmin hatası: {str(e)}")

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = BitcoinPredictorApp()

    window.show()

    sys.exit(app.exec\_())

şimdi de açıklamalı kod:

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

Gpt apilerini ekleme şeysini unutma.

Algoritma

import sys

import pandas as pd

import yfinance as yf

import numpy as np

from datetime import datetime, timedelta

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,

                           QPushButton, QLabel, QLineEdit, QMessageBox,

                           QTextEdit, QSplitter, QDialog)

from PyQt5.QtCore import Qt

class AnalysisDialog(QDialog):

    def \_\_init\_\_(self, analysis\_text, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Tahmin Analizi")

        self.setGeometry(200, 200, 600, 400)

        layout = QVBoxLayout()

        text\_edit = QTextEdit()

        text\_edit.setReadOnly(True)

        text\_edit.setPlainText(analysis\_text)

        layout.addWidget(text\_edit)

        close\_button = QPushButton("Kapat")

        close\_button.clicked.connect(self.close)

        layout.addWidget(close\_button)

        self.setLayout(layout)

class BitcoinPredictorApp(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Bitcoin Tahmin Uygulaması")

        self.setGeometry(100, 100, 1000, 800)

        main\_widget = QWidget()

        self.setCentralWidget(main\_widget)

        main\_layout = QVBoxLayout(main\_widget)

        splitter = QSplitter(Qt.Horizontal)

        left\_widget = QWidget()

        left\_layout = QVBoxLayout(left\_widget)

        self.figure, self.ax = plt.subplots(figsize=(8, 6))

        self.canvas = FigureCanvas(self.figure)

        left\_layout.addWidget(self.canvas)

        date\_widget = QWidget()

        date\_layout = QVBoxLayout(date\_widget)

        self.date\_label = QLabel("Tahmin Tarihi (YYYY-MM-DD):")

        self.date\_input = QLineEdit()

        self.date\_input.setPlaceholderText("Örnek: 2024-12-31")

        date\_layout.addWidget(self.date\_label)

        date\_layout.addWidget(self.date\_input)

        self.predict\_button = QPushButton("Tahmin Et")

        self.predict\_button.clicked.connect(self.make\_prediction)

        date\_layout.addWidget(self.predict\_button)

        left\_layout.addWidget(date\_widget)

        self.results\_text = QTextEdit()

        self.results\_text.setReadOnly(True)

        self.results\_text.setMinimumWidth(300)

        splitter.addWidget(left\_widget)

        splitter.addWidget(self.results\_text)

        main\_layout.addWidget(splitter)

        self.prepare\_data\_and\_model()

        self.plot\_bitcoin\_data()

    def prepare\_data\_and\_model(self):

        self.ticker = "BTC-USD"

        start\_date = "2024-04-01"

        end\_date = datetime.now().strftime("%Y-%m-%d")

        try:

            self.df = yf.download(self.ticker, start=start\_date, end=end\_date)

            self.df.loc[:, 'preds'] = self.df['Close'].shift(-1)

            self.df.dropna(inplace=True)

            X = self.df[['Open', 'High', 'Low', 'Volume']]

            y = self.df['preds']

            self.scaler = StandardScaler()

            X\_scaled = self.scaler.fit\_transform(X)

            self.model = LinearRegression()

            self.model.fit(X\_scaled, y)

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Veri hazırlama hatası: {str(e)}")

    def plot\_bitcoin\_data(self):

        try:

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax, label='Gerçek Veriler', linewidth=2)

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği')

            self.ax.set\_xlabel('Tarih')

            self.ax.set\_ylabel('Fiyat (USD)')

            self.ax.grid(True)

            self.ax.legend()

            self.canvas.draw()

        except Exception as e:

            QMessageBox.warning(self, "Uyarı", f"Grafik çizim hatası: {str(e)}")

    def calculate\_accuracy\_metrics(self, actual, predicted):

        mape = np.mean(np.abs((actual - predicted) / actual)) \* 100

        rmse = np.sqrt(np.mean((actual - predicted) \*\* 2))

        accuracy = 100 - mape

        analysis\_text = f"""Tahmin Analizi Detayları:

1. Genel Doğruluk Oranı: {accuracy:.2f}%

2. Ortalama Mutlak Yüzde Hata (MAPE): {mape:.2f}%

3. Kök Ortalama Kare Hata (RMSE): ${rmse:.2f}

4. Toplam Tahmin Sayısı: {len(actual)}

Tahmin Başarısı Dağılımı:

- Çok İyi Tahminler (Hata < 2%): {sum(np.abs((actual - predicted) / actual) < 0.02)} adet

- İyi Tahminler (Hata < 5%): {sum(np.abs((actual - predicted) / actual) < 0.05)} adet

- Orta Tahminler (Hata < 10%): {sum(np.abs((actual - predicted) / actual) < 0.10)} adet

- Zayıf Tahminler (Hata >= 10%): {sum(np.abs((actual - predicted) / actual) >= 0.10)} adet

Tahmin Modeli Hakkında:

- Model tipi: Lineer Regresyon

- Kullanılan özellikler: Açılış Fiyatı, En Yüksek, En Düşük, İşlem Hacmi

- Eğitim verisi: Son 6 aylık Bitcoin fiyat verileri

Not: Kripto para piyasaları yüksek volatiliteye sahiptir.

Bu tahminler sadece geçmiş verilere dayalı istatistiksel bir analiz sunar."""

        return accuracy, analysis\_text

    def predict\_next\_days(self, end\_date):

        last\_real\_date = self.df.index[-1].tz\_localize(None)

        target\_date = datetime.strptime(end\_date, '%Y-%m-%d')

        days\_to\_predict = (target\_date - last\_real\_date).days

        if days\_to\_predict <= 0:

            # Geçmiş tarih analizi

            start\_date = (target\_date - timedelta(days=180)).strftime('%Y-%m-%d')

            end\_date = target\_date.strftime('%Y-%m-%d')

            try:

                df\_past = yf.download(self.ticker, start=start\_date, end=end\_date)

                df\_past.loc[:, 'preds'] = df\_past['Close'].shift(-1)

                df\_past.dropna(inplace=True)

                X\_past = df\_past[['Open', 'High', 'Low', 'Volume']]

                actual\_prices = df\_past['Close']

                X\_past\_scaled = self.scaler.transform(X\_past)

                predicted\_prices = self.model.predict(X\_past\_scaled)

                accuracy, analysis\_text = self.calculate\_accuracy\_metrics(actual\_prices.values, predicted\_prices)

                # Grafiği güncelle

                self.ax.clear()

                self.ax.plot(df\_past.index, actual\_prices, label='Gerçek Veriler', linewidth=2)

                self.ax.plot(df\_past.index, predicted\_prices, '--', label=f'Model Tahminleri (Doğruluk: {accuracy:.2f}%)', linewidth=2)

                self.ax.set\_title('Bitcoin (BTC) Geçmiş Fiyat Analizi')

                self.ax.legend()

                self.ax.grid(True)

                self.canvas.draw()

                # Analiz penceresini göster

                dialog = AnalysisDialog(analysis\_text, self)

                dialog.exec\_()

                return pd.DataFrame()

            except Exception as e:

                QMessageBox.critical(self, "Hata", f"Geçmiş veri tahmini hatası: {str(e)}")

                return pd.DataFrame()

        # Gelecek tahminleri

        predictions = []

        dates = []

        current\_features = self.df[['Open', 'High', 'Low', 'Volume']].iloc[-1:].values

        for i in range(days\_to\_predict):

            current\_scaled = self.scaler.transform(current\_features)

            next\_price = self.model.predict(current\_scaled)[0]

            next\_date = last\_real\_date + timedelta(days=i+1)

            predictions.append(next\_price)

            dates.append(next\_date)

            current\_features = np.array([[

                float(next\_price),

                float(next\_price \* 1.01),

                float(next\_price \* 0.99),

                float(current\_features[0][3])

            ]])

        future\_df = pd.DataFrame({

            'Tarih': dates,

            'Tahmin': predictions

        })

        # Tahmin analizi metnini oluştur

        analysis\_text = f"""Gelecek Tahmin Analizi:

1. Tahmin Edilen Gün Sayısı: {len(predictions)}

2. Başlangıç Tarihi: {dates[0].strftime('%Y-%m-%d')}

3. Bitiş Tarihi: {dates[-1].strftime('%Y-%m-%d')}

4. Başlangıç Fiyatı: ${predictions[0]:.2f}

5. Tahmin Edilen Son Fiyat: ${predictions[-1]:.2f}

6. Toplam Değişim: {((predictions[-1] - predictions[0]) / predictions[0] \* 100):.2f}%

Tahmin Modeli Hakkında:

- Kullanılan model: Lineer Regresyon

- Eğitim verisi: {self.df.index[0].strftime('%Y-%m-%d')} - {self.df.index[-1].strftime('%Y-%m-%d')}

- Kullanılan özellikler: Açılış, En Yüksek, En Düşük, İşlem Hacmi

Not: Bu tahminler geçmiş verilere dayalı istatistiksel bir projeksiyon sunar.

Kripto para piyasaları yüksek volatiliteye sahiptir ve beklenmedik

olaylar fiyatları önemli ölçüde etkileyebilir."""

        # Analiz penceresini göster

        dialog = AnalysisDialog(analysis\_text, self)

        dialog.exec\_()

        return future\_df

    def make\_prediction(self):

        try:

            end\_date = self.date\_input.text()

            datetime.strptime(end\_date, '%Y-%m-%d')

            predictions\_df = self.predict\_next\_days(end\_date)

            if predictions\_df.empty:

                return

            # Grafiği güncelle

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax, label='Gerçek Veriler', linewidth=2)

            predictions\_df.set\_index('Tarih')['Tahmin'].plot(ax=self.ax,

                                                           label='Gelecek Tahminleri',

                                                           style='--',

                                                           linewidth=2)

            self.ax.legend()

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği ve Tahminler')

            self.ax.grid(True)

            self.canvas.draw()

            # Sonuçları metin alanına yaz

            self.results\_text.clear()

            self.results\_text.append("Tahmin Sonuçları:\n")

            for \_, row in predictions\_df.iterrows():

                self.results\_text.append(

                    f"Tarih: {row['Tarih'].strftime('%Y-%m-%d')}\n"

                    f"Tahmin: ${row['Tahmin']:.2f}\n\n"

                )

        except ValueError:

            QMessageBox.warning(self, "Uyarı", "Geçersiz tarih formatı! YYYY-MM-DD formatında giriniz.")

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Tahmin hatası: {str(e)}")

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = BitcoinPredictorApp()

    window.show()

    sys.exit(app.exec\_())

kodum bu

ve tarih yazınca fiyatı tahmin ediyor. peki bunu nasıl yapıyor, hangi algoritma ile, tahmin edilen fiyat nasıl oluşuyor?

Bu uygulama, kullanıcının belirttiği bir tarihteki Bitcoin fiyatını tahmin etmek için bir *lineer regresyon modeli* kullanarak geçmiş fiyat verilerini analiz ediyor. İşleyişi ve tahmin sürecini adım adım inceleyelim:

**1. Veri Hazırlığı**

Uygulama, geçmiş 6 aylık Bitcoin verilerini Yahoo Finance üzerinden indiriyor. Bu veriler:

* Açılış fiyatı (Open)
* Gün içindeki en yüksek fiyat (High)
* Gün içindeki en düşük fiyat (Low)
* İşlem hacmi (Volume)
* Kapanış fiyatı (Close) sütunlarından oluşuyor.

Ardından, kapanış fiyatını bir gün kaydırarak (shift(-1)) 'preds' adında bir sütun oluşturuyor. Bu sütun, bir sonraki günün kapanış fiyatını hedef (y) değişkeni olarak belirliyor. Bu şekilde, model bir gün sonrasının kapanış fiyatını tahmin etmek için eğitiliyor.

**2. Özelliklerin Standartlaştırılması**

Makine öğrenimi algoritmalarının daha iyi performans göstermesi için *StandardScaler* kullanılarak özellikler (Open, High, Low, Volume) standartlaştırılıyor. Standartlaştırma, verilerin ortalamasını sıfıra, standart sapmasını ise bire eşitleyerek özelliklerin aynı ölçekte olmasını sağlıyor.

**3. Model Eğitimi**

* **Model**: *Lineer Regresyon Modeli*
* **Özellikler**: Açılış fiyatı, en yüksek fiyat, en düşük fiyat, işlem hacmi

Lineer regresyon modeli, bu özelliklerle bir gün sonrasının kapanış fiyatını tahmin etmek için eğitiliyor. Model, eğitim sırasında her bir özelliğe ağırlıklar vererek bu özelliklerin kapanış fiyatı üzerindeki etkisini öğreniyor.

**4. Tahmin Süreci**

Kullanıcı bir tahmin tarihi belirttiğinde, uygulama bu tarihi analiz ediyor:

* Eğer tarih geçmişteyse, yani son güncel verinin tarihinden önce bir tarihse, uygulama geçmiş verilerle doğruluk hesaplaması yaparak geçmiş fiyat tahminini analiz eder.
* Eğer tarih gelecekteyse, model bir sonraki günün fiyatını tahmin ederek ardışık tahminler üretir. Her bir gün için:
  + **Özellik Değerlerinin Güncellenmesi**: Tahmin edilen fiyat Open, High, Low özelliklerinde kullanılır. High tahmin fiyatının %1 üzeri, Low ise %1 altı olarak varsayılır.
  + Yeni tahmin edilen fiyat, bir sonraki günün başlangıç noktası olarak modelin girdisi olur.

**5. Tahmin Doğruluğu ve Analiz**

Modelin doğruluğunu ölçmek için şu metrikler kullanılıyor:

* **MAPE (Ortalama Mutlak Yüzde Hata)**: Tahmin edilen değerlerin gerçek değerlere ne kadar yakın olduğunu yüzdesel olarak gösterir.
* **RMSE (Kök Ortalama Kare Hata)**: Tahminlerdeki hatanın ortalama büyüklüğünü gösterir.
* **Genel Doğruluk Oranı**: %100 üzerinden MAPE'yi çıkararak modelin ne kadar başarılı olduğunu belirtir.

**Sonuç**

Bu uygulama, geçmiş verilerden öğrenilen ilişkileri kullanarak basit bir lineer regresyon modeli ile Bitcoin fiyatını tahmin ediyor. Ancak lineer regresyon, verilerin doğrusal bir yapıda olduğunu varsaydığından, fiyatın doğrusal olmayan hareketlerini doğru tahmin etmekte sınırlı olabilir.

**Şunu dedin:**

MAPE nedir

**MAPE** (Mean Absolute Percentage Error) veya Türkçe adıyla *Ortalama Mutlak Yüzde Hata*, bir tahmin modelinin performansını değerlendirmek için kullanılan bir hata metriğidir. MAPE, tahmin edilen değerlerin gerçek değerlere ne kadar yakın olduğunu yüzdesel olarak ifade eder ve modelin doğruluğunu anlamak için önemli bir ölçüttür.

metin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim

Açıklama otomatik olarak oluşturuldu

Daha özet:

Kod, Bitcoin fiyatını tahmin etmek için **Lineer Regresyon** algoritmasını kullanıyor. Geçmiş verilere bakarak (açılış, en yüksek, en düşük fiyatlar ve hacim gibi özelliklerle), bir gün sonrasının kapanış fiyatını tahmin etmeyi öğreniyor. Kullanıcı bir tarih girdiğinde, model bugünkü fiyat verilerini kullanarak her gün için ardışık tahminler yapıyor; tahmin edilen fiyatlar bir sonraki günün giriş verileri olarak kullanılıyor. Böylece belirtilen tarihe kadar fiyat projeksiyonu yapılıyor

pip install xgboost

yaptık.

XGBoost uzun vadeli tahminler için iyi bir seçim olabilir. Çünkü:

1. Doğrusal olmayan ilişkileri yakalayabilir
2. Aşırı öğrenmeye karşı dayanıklıdır
3. Birden fazla özelliği etkili şekilde kullanabilir
4. Zaman serisi tahminlerinde başarılı performans gösterir

XGBoost'un avantajları:

1. Doğrusal olmayan ilişkileri yakalayabilir
2. Overfitting'e karşı dayanıklı
3. Otomatik özellik seçimi yapabilir
4. Trend değişimlerini daha iyi tahmin edebilir

Dezavantajları:

1. Daha fazla hesaplama gücü gerektirir
2. Linear Regression'a göre daha karmaşık
3. Parametre ayarı gerektirir

 İlk 7 gün için:

* Linear Regression: %70 ağırlık
* XGBoost: %30 ağırlık

 7 günden sonrası için:

* Linear Regression: %30 ağırlık
* XGBoost: %70 ağırlık

Teknik Özellikler:

* Her tahmin sonrası tüm teknik indikatörler güncellenir
* Gelecek tahminler için bu güncel veriler kullanılır

Windows 10 sdk yükledim.

metin, ekran görüntüsü, yazı tipi, cebir içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ltsm iyileştirme

import sys

import pandas as pd

import yfinance as yf

import numpy as np

from datetime import datetime, timedelta

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from PyQt5.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,

                           QPushButton, QLabel, QLineEdit, QMessageBox,

                           QTextEdit, QSplitter, QDialog)

from PyQt5.QtCore import Qt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf

from tensorflow.keras.layers import LSTM, Dropout, Dense

class AnalysisDialog(QDialog):

    def \_\_init\_\_(self, analysis\_text, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Tahmin Analizi")

        self.setGeometry(200, 200, 800, 600)

        layout = QVBoxLayout()

        text\_edit = QTextEdit()

        text\_edit.setReadOnly(True)

        text\_edit.setPlainText(analysis\_text)

        layout.addWidget(text\_edit)

        close\_button = QPushButton("Kapat")

        close\_button.clicked.connect(self.close)

        layout.addWidget(close\_button)

        self.setLayout(layout)

class BitcoinPredictorApp(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Bitcoin Tahmin Uygulaması")

        self.setGeometry(100, 100, 1200, 800)

        main\_widget = QWidget()

        self.setCentralWidget(main\_widget)

        main\_layout = QVBoxLayout(main\_widget)

        splitter = QSplitter(Qt.Horizontal)

        left\_widget = QWidget()

        left\_layout = QVBoxLayout(left\_widget)

        self.figure, self.ax = plt.subplots(figsize=(10, 6))

        self.canvas = FigureCanvas(self.figure)

        left\_layout.addWidget(self.canvas)

        date\_widget = QWidget()

        date\_layout = QVBoxLayout(date\_widget)

        self.date\_label = QLabel("Tahmin Tarihi (YYYY-MM-DD):")

        self.date\_input = QLineEdit()

        self.date\_input.setPlaceholderText("Örnek: 2024-12-31")

        date\_layout.addWidget(self.date\_label)

        date\_layout.addWidget(self.date\_input)

        self.predict\_button = QPushButton("Tahmin Et")

        self.predict\_button.clicked.connect(self.make\_prediction)

        date\_layout.addWidget(self.predict\_button)

        left\_layout.addWidget(date\_widget)

        self.results\_text = QTextEdit()

        self.results\_text.setReadOnly(True)

        self.results\_text.setMinimumWidth(300)

        splitter.addWidget(left\_widget)

        splitter.addWidget(self.results\_text)

        main\_layout.addWidget(splitter)

        self.prepare\_data\_and\_models()

        self.plot\_bitcoin\_data()

    def prepare\_data\_and\_models(self):

        """Veri hazırlama ve model eğitimi"""

        self.ticker = "BTC-USD"

        # 4 yıllık veri al

        start\_date = (datetime.now() - timedelta(days=4\*365)).strftime('%Y-%m-%d')

        end\_date = datetime.now().strftime('%Y-%m-%d')

        try:

            # Verileri indir

            self.df = yf.download(self.ticker, start=start\_date, end=end\_date)

            # Teknik indikatörler ekle

            self.add\_technical\_indicators()

            # Veriyi eğitim için hazırla

            self.prepare\_training\_data()

            # Modelleri eğit

            self.train\_models()

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Veri hazırlama hatası: {str(e)}")

    def add\_technical\_indicators(self):

        """Teknik indikatörleri hesapla"""

        try:

            # Returns ve volatilite

            self.df['Returns'] = self.df['Close'].pct\_change()

            self.df['Volatility'] = self.df['Returns'].rolling(window=20).std()

            # Hareketli ortalamalar

            self.df['MA7'] = self.df['Close'].rolling(window=7).mean()

            self.df['MA30'] = self.df['Close'].rolling(window=30).mean()

            self.df['MA90'] = self.df['Close'].rolling(window=90).mean()

            self.df['MA200'] = self.df['Close'].rolling(window=200).mean()

            # RSI

            self.df['RSI'] = self.calculate\_rsi(self.df['Close'])

            # MACD

            exp1 = self.df['Close'].ewm(span=12, adjust=False).mean()

            exp2 = self.df['Close'].ewm(span=26, adjust=False).mean()

            self.df['MACD'] = exp1 - exp2

            self.df['Signal\_Line'] = self.df['MACD'].ewm(span=9, adjust=False).mean()

            # Bollinger Bands düzeltilmiş hesaplama

            bb\_window = 20

            rolling\_mean = self.df['Close'].rolling(window=bb\_window).mean()

            rolling\_std = self.df['Close'].rolling(window=bb\_window).std()

            self.df['BB\_middle'] = rolling\_mean

            self.df['BB\_upper'] = rolling\_mean + (rolling\_std \* 2)

            self.df['BB\_lower'] = rolling\_mean - (rolling\_std \* 2)

            # Eksik değerleri temizle

            self.df.dropna(inplace=True)

        except Exception as e:

            print(f"Teknik indikatör hesaplama hatası: {str(e)}")

            raise e

    def calculate\_new\_features(self, predictions, current\_price):

        """Yeni tahmin için özellikleri hesapla"""

        recent\_prices = predictions[-7:] if len(predictions) >= 7 else predictions + [current\_price]

        features = {

            'Open': current\_price,

            'High': current\_price \* (1 + np.random.uniform(0, 0.02)),

            'Low': current\_price \* (1 - np.random.uniform(0, 0.02)),

            'Close': current\_price,

            'Volume': self.df['Volume'].mean() \* (1 + np.random.uniform(-0.2, 0.2)),

            'Returns': (current\_price - predictions[-1])/predictions[-1] if predictions else 0,

            'Volatility': np.std(recent\_prices) if len(recent\_prices) > 1 else 0,

            'MA7': np.mean(recent\_prices),

            'MA30': current\_price,  # Basitleştirilmiş

            'MA90': current\_price,  # Basitleştirilmiş

            'MA200': current\_price,  # Basitleştirilmiş

            'RSI': 50 + np.random.uniform(-10, 10),  # Rastgele RSI

            'MACD': 0,  # Basitleştirilmiş

            'Signal\_Line': 0,  # Basitleştirilmiş

            'BB\_middle': current\_price,

            'BB\_upper': current\_price \* 1.02,

            'BB\_lower': current\_price \* 0.98

        }

        return list(features.values())

    def calculate\_rsi(self, prices, period=14):

        """RSI hesapla"""

        delta = prices.diff()

        gain = (delta.where(delta > 0, 0)).rolling(window=period).mean()

        loss = (-delta.where(delta < 0, 0)).rolling(window=period).mean()

        rs = gain / loss

        return 100 - (100 / (1 + rs))

    def prepare\_training\_data(self):

        """Eğitim verilerini hazırla"""

        # LSTM için özellikler - daha fazla özellik ekledik

        self.features = ['Open', 'High', 'Low', 'Close', 'Volume', 'Returns',

                        'Volatility', 'MA7', 'MA30', 'MA90', 'MA200',

                        'RSI', 'MACD', 'Signal\_Line',

                        'BB\_middle', 'BB\_upper', 'BB\_lower']

        # Veriyi ölçeklendir

        self.scaler = MinMaxScaler(feature\_range=(0, 1))

        self.price\_scaler = MinMaxScaler(feature\_range=(0, 1))

        # Tüm özellikleri ölçeklendir

        self.X\_scaled = self.scaler.fit\_transform(self.df[self.features])

        self.y\_scaled = self.price\_scaler.fit\_transform(self.df[['Close']])

        # LSTM için veri hazırlığı

        self.prepare\_lstm\_sequences()

    def prepare\_lstm\_training\_data(self, time\_step=60):

        """LSTM için zaman pencereli eğitim verilerini hazırla"""

        self.X\_lstm = []

        self.y\_lstm = []

        for i in range(time\_step, len(self.y\_scaled)):

            self.X\_lstm.append(self.y\_scaled[i-time\_step:i, 0])

            self.y\_lstm.append(self.y\_scaled[i, 0])

        # LSTM için veriyi numpy array olarak kaydedin

        self.X\_lstm, self.y\_lstm = np.array(self.X\_lstm), np.array(self.y\_lstm)

        # LSTM giriş şekli (ornek, zaman adımı, özellik sayısı) - burada özellik sayısı 1

        self.X\_lstm = np.reshape(self.X\_lstm, (self.X\_lstm.shape[0], self.X\_lstm.shape[1], 1))

    def prepare\_lstm\_sequences(self, time\_step=60):

        """LSTM için çok değişkenli zaman serisi verisi hazırla"""

        self.X\_lstm = []

        self.y\_lstm = []

        # Tüm özellikler için sekanslar oluştur

        for i in range(time\_step, len(self.X\_scaled)):

            self.X\_lstm.append(self.X\_scaled[i-time\_step:i])

            # Sadece kapanış fiyatını hedef olarak al

            self.y\_lstm.append(self.df['Close'].iloc[i])

        self.X\_lstm = np.array(self.X\_lstm)

        # y\_lstm'i doğru şekilde ölçeklendir

        self.y\_lstm = self.price\_scaler.fit\_transform(np.array(self.y\_lstm).reshape(-1, 1))

        print(f"X\_lstm shape: {self.X\_lstm.shape}")

        print(f"y\_lstm shape: {self.y\_lstm.shape}")

    def build\_lstm\_model(self):

        """Geliştirilmiş LSTM modeli"""

        model = Sequential([

            LSTM(100, return\_sequences=True, input\_shape=(self.X\_lstm.shape[1], self.X\_lstm.shape[2])),

            Dropout(0.2),

            LSTM(100, return\_sequences=True),

            Dropout(0.2),

            LSTM(50),

            Dropout(0.2),

            Dense(25),

            Dense(1)

        ])

        model.compile(optimizer='adam', loss='huber', metrics=['mae'])

        return model

    def train\_models(self):

        """Modeli eğit"""

        self.lstm\_model = self.build\_lstm\_model()

        # Early stopping ve model checkpoint ekle

        early\_stopping = tf.keras.callbacks.EarlyStopping(

            monitor='val\_loss',

            patience=5,  # 15'ten 5'e düşürüldü

            restore\_best\_weights=True

        )

        # Öğrenme oranı düşürme

        reduce\_lr = tf.keras.callbacks.ReduceLROnPlateau(

            monitor='val\_loss',

            factor=0.2,

            patience=3,  # 5'ten 3'e düşürüldü

            min\_lr=0.0001

        )

        # Modeli eğit

        history = self.lstm\_model.fit(

            self.X\_lstm, self.y\_lstm,

            epochs=30,  # 200'den 30'a düşürüldü

            batch\_size=16,

            validation\_split=0.2,

            callbacks=[early\_stopping, reduce\_lr],

            verbose=1

        )

    def plot\_bitcoin\_data(self):

        """Bitcoin verilerini göster"""

        try:

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax, label='Bitcoin Fiyatı', linewidth=2)

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği')

            self.ax.set\_xlabel('Tarih')

            self.ax.set\_ylabel('Fiyat (USD)')

            self.ax.grid(True)

            self.ax.legend()

            self.canvas.draw()

        except Exception as e:

            QMessageBox.warning(self, "Uyarı", f"Grafik çizim hatası: {str(e)}")

    def make\_prediction(self):

        """Tahmin yap"""

        try:

            end\_date = self.date\_input.text()

            datetime.strptime(end\_date, '%Y-%m-%d')

            predictions\_df = self.predict\_next\_days(end\_date)

            if predictions\_df.empty:

                return

            # Grafiği güncelle

            self.ax.clear()

            self.df['Close'].plot(ax=self.ax, label='Gerçek Veriler', linewidth=2)

            predictions\_df.set\_index('Tarih')['Tahmin'].plot(ax=self.ax,

                                                           label='Tahminler',

                                                           style='--',

                                                           linewidth=2)

            self.ax.legend()

            self.ax.set\_title('Bitcoin (BTC) Fiyat Grafiği ve Tahminler')

            self.ax.grid(True)

            self.canvas.draw()

            # Sonuçları göster

            self.results\_text.clear()

            self.results\_text.append("Tahmin Sonuçları:\n")

            for \_, row in predictions\_df.iterrows():

                self.results\_text.append(

                    f"Tarih: {row['Tarih'].strftime('%Y-%m-%d')}\n"

                    f"Tahmin: ${row['Tahmin']:.2f}\n\n"

                )

        except ValueError:

            QMessageBox.warning(self, "Uyarı", "Geçersiz tarih formatı! YYYY-MM-DD formatında giriniz.")

        except Exception as e:

            QMessageBox.critical(self, "Hata", f"Tahmin hatası: {str(e)}")

    def predict\_next\_days(self, end\_date):

        """Gelecek günleri tahmin et"""

        last\_real\_date = self.df.index[-1].tz\_localize(None)

        target\_date = datetime.strptime(end\_date, '%Y-%m-%d')

        days\_to\_predict = (target\_date - last\_real\_date).days

        if days\_to\_predict <= 0:

            return self.analyze\_past\_date(target\_date)

        else:

            return self.predict\_future\_date(days\_to\_predict, last\_real\_date)

    def predict\_future\_date(self, days\_to\_predict, last\_real\_date):

        """Geliştirilmiş gelecek tarih tahmini"""

        try:

            predictions = []

            dates = []

            # Son 60 günlük veriyi al

            last\_sequence = self.X\_scaled[-60:]

            print("Başlangıç sequence shape:", last\_sequence.shape)

            for i in range(days\_to\_predict):

                # LSTM için giriş verisini hazırla

                lstm\_input = last\_sequence.reshape(1, 60, self.X\_scaled.shape[1])

                print(f"Input shape: {lstm\_input.shape}")

                # Tahmin yap

                pred\_scaled = self.lstm\_model.predict(lstm\_input, verbose=0)

                print(f"Raw prediction shape: {pred\_scaled.shape}")

                # Tahmin değerini kapanış fiyatına dönüştür

                pred\_scaled\_reshaped = np.array([[float(pred\_scaled[0][0])]])

                pred\_price = self.price\_scaler.inverse\_transform(pred\_scaled\_reshaped)[0][0]

                print(f"Predicted price: {pred\_price}")

                # Rastgelelik ekle

                volatility = np.std(self.df['Returns'].iloc[-30:]) \* 300  # Artırıldı

                random\_change = np.random.normal(0, volatility)

                pred\_price \*= (1 + random\_change/100)

                next\_date = last\_real\_date + timedelta(days=i+1)

                predictions.append(pred\_price)

                dates.append(next\_date)

                # Yeni özellikleri hesapla ve numpy dizisine dönüştür

                features\_dict = {

                    'Open': pred\_price,

                    'High': pred\_price \* (1 + np.random.uniform(0, 0.02)),

                    'Low': pred\_price \* (1 - np.random.uniform(0, 0.02)),

                    'Close': pred\_price,

                    'Volume': float(self.df['Volume'].mean() \* (1 + np.random.uniform(-0.2, 0.2))),

                    'Returns': float((pred\_price - predictions[-1])/predictions[-1] if len(predictions) > 0 else 0),

                    'Volatility': float(np.std(predictions[-7:]) if len(predictions) > 1 else 0),

                    'MA7': float(np.mean(predictions[-7:]) if len(predictions) >= 7 else pred\_price),

                    'MA30': float(np.mean(predictions[-30:]) if len(predictions) >= 30 else pred\_price),

                    'MA90': float(pred\_price),

                    'MA200': float(pred\_price),

                    'RSI': float(50 + np.random.uniform(-10, 10)),

                    'MACD': float(0),

                    'Signal\_Line': float(0),

                    'BB\_middle': float(pred\_price),

                    'BB\_upper': float(pred\_price \* 1.02),

                    'BB\_lower': float(pred\_price \* 0.98)

                }

                # Sözlüğü sıralı bir listeye dönüştür

                new\_features = [features\_dict[feature] for feature in self.features]

                new\_features = np.array(new\_features, dtype=np.float32).reshape(1, -1)

                # Son sequence'i güncelle

                new\_row\_scaled = self.scaler.transform(new\_features)

                last\_sequence = np.vstack((last\_sequence[1:], new\_row\_scaled[0]))

            df = pd.DataFrame({

                'Tarih': dates,

                'Tahmin': predictions

            })

            print("Tahminler tamamlandı:", len(predictions), "gün")

            return df

        except Exception as e:

            print(f"Hata detayı: {str(e)}")

            print(f"Stack trace:")

            import traceback

            traceback.print\_exc()

            QMessageBox.critical(self, "Hata", f"Gelecek tahmini hatası: {str(e)}")

            return pd.DataFrame()

    def update\_technical\_features(self, current\_data, next\_price, predictions):

        """Bir sonraki gün için teknik özellikleri güncelle"""

        current\_data['Close'] = next\_price

        current\_data['Open'] = next\_price

        current\_data['High'] = next\_price \* 1.02

        current\_data['Low'] = next\_price \* 0.98

        # Son n günlük tahminleri al

        recent\_predictions = (predictions[-30:] if len(predictions) > 0

                            else [next\_price])

        # Teknik indikatörleri güncelle

        current\_data['Returns'] = (next\_price - predictions[-1]) / predictions[-1] if predictions else 0

        current\_data['Volatility'] = np.std(recent\_predictions) if len(recent\_predictions) > 1 else 0

        current\_data['MA7'] = np.mean(recent\_predictions[-7:]) if len(recent\_predictions) >= 7 else next\_price

        current\_data['MA30'] = np.mean(recent\_predictions[-30:]) if len(recent\_predictions) >= 30 else next\_price

        current\_data['MA90'] = next\_price  # Yeterli veri olmadığı için basitleştirildi

        current\_data['MA200'] = next\_price  # Yeterli veri olmadığı için basitleştirildi

        # RSI ve diğer indikatörler için basitleştirilmiş güncellemeler

        current\_data['RSI'] = 50  # Nötr değer

        current\_data['MACD'] = 0

        current\_data['Signal\_Line'] = 0

        current\_data['BB\_middle'] = next\_price

        current\_data['BB\_upper'] = next\_price \* 1.02

        current\_data['BB\_lower'] = next\_price \* 0.98

    def analyze\_past\_date(self, target\_date):

        """Geçmiş tarih analizi"""

        try:

            # Hedef tarihe en yakın tarihi bul

            closest\_date = self.df.index[self.df.index.get\_indexer([target\_date], method='nearest')[0]]

            # Gerçek değeri al

            actual\_price = self.df.loc[closest\_date, 'Close']

            # DataFrame oluştur

            return pd.DataFrame({

                'Tarih': [closest\_date],

                'Tahmin': [actual\_price]

            })

        except Exception as e:

            print(f"Geçmiş tarih analizi hatası: {str(e)}")

            return pd.DataFrame()

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = BitcoinPredictorApp()

    window.show()

    sys.exit(app.exec\_())

Kodunuzda rastgele şok olayları (%10 olasılıkla büyük hareketler) predict\_future\_date fonksiyonunda tanımlanmıştır. Bu bölümde, her bir tahmin adımı için şok olaylarının nasıl oluşturulduğunu görebilirsiniz:

python

Kodu kopyala

# Rastgele şoklar (düşük olasılıklı büyük hareketler)

shock = 0

if np.random.random() < 0.1: # %10 olasılıkla

shock = np.random.normal(0, short\_term\_volatility \* 2)

**2. Validation Split ve Early Stopping Kullanımı**

* validation\_split eğitim sırasında verilerin bir kısmını doğrulama (validation) olarak ayırarak modelin sadece eğitim verilerine değil, aynı zamanda doğrulama verilerine de uygun hale gelmesini sağlar. Böylece aşırı öğrenme ihtimali azalır.
* early stopping, modelin doğrulama kaybı (val\_loss) belli bir süre boyunca iyileşmediğinde eğitimi durdurur. Bu da modelin aşırı uyum sağlamasını (overfitting) önler.

Büyük düşüş olmasın(oldu ama daha istikrarlı zikzak olabilir kodunun yapay zeka açıklaması:

Bu kodda çeşitli yapay zeka ve makine öğrenmesi modelleri kullanılmıştır. İşte detaylı analiz:

1. LSTM (Long Short-Term Memory) Neural Network:

* build\_lstm\_model() fonksiyonunda LSTM modeli oluşturuluyor
* Derin öğrenme tabanlı bir model
* Zaman serisi tahminleri için özellikle etkili
* Model mimarisi:
  + 4 LSTM katmanı (150, 100, 100, 50 nöron)
  + Dropout katmanları (0.3, 0.3, 0.3, 0.2)
  + 2 Dense katman (50, 25 nöron)
  + Final Dense katman (1 nöron)

1. Ensemble Model (EnsembleModel sınıfı): Üç farklı model bir arada kullanılıyor:

* Random Forest (RandomForestRegressor)
  + Ağaç tabanlı bir topluluk öğrenme algoritması
  + 100 ağaçlı orman kullanılıyor
* XGBoost (XGBRegressor)
  + Gradient boosting tabanlı güçlü bir algoritma
  + Ağaç tabanlı modellerin gelişmiş versiyonu
* LightGBM (LGBMRegressor)
  + Microsoft'un geliştirdiği hızlı gradient boosting framework'ü
  + Ağaç tabanlı bir model

1. Modellerin Kullanıldığı Önemli Fonksiyonlar:

def train\_models(self):

# Tüm modellerin eğitimini başlatıyor

# LSTM ve ensemble modelleri burada eğitiliyor

def make\_prediction(self):

# Tahmin yapma işlemlerini yönetiyor

# Tüm modellerin tahminlerini kullanıyor

def predict\_future\_date(self):

# Gelecek tarihler için tahmin yapıyor

# LSTM ve ensemble modellerinin tahminlerini kombine ediyor

1. Özel Özellikler:

* Custom Loss Function: LSTM modelinde özel bir kayıp fonksiyonu kullanılıyor

def custom\_loss(y\_true, y\_pred):

mse = tf.reduce\_mean(tf.square(y\_true - y\_pred))

decline\_penalty = tf.reduce\_mean(tf.maximum(0., y\_true - y\_pred) \* 2)

return mse + decline\_penalty

Veri İşleme ve Hazırlama:

def prepare\_training\_data(self):

# Veriyi ölçeklendirme ve modeller için hazırlama

def prepare\_lstm\_sequences(self):

# LSTM için özel veri hazırlığı

Bu modeller birlikte çalışarak:

* Bitcoin fiyat tahminleri yapıyor
* Trend analizleri gerçekleştiriyor
* Farklı zaman dilimlerinde tahminler üretiyor
* Teknik göstergeleri hesaplıyor

Modeller özellikle zaman serisi analizi ve finansal tahmin için optimize edilmiş durumda. LSTM modeli uzun vadeli bağımlılıkları öğrenirken, ensemble modeller farklı özelliklerdeki patternleri yakalıyor.

Kodda kullanılan yapay zeka modelleri özet olarak şunlardır:

1. LSTM (Long Short-Term Memory Neural Network)
   * Derin öğrenme / yapay sinir ağı modeli
   * Ana tahmin modeli olarak kullanılıyor
2. Ensemble (Topluluk) Modelleri:
   * Random Forest
   * XGBoost
   * LightGBM

YOLOv3 nedir? – Tanım YOLO

YOLOv3 (You Only Look Once, Sürüm 3) , videolarda, canlı yayınlarda veya görüntülerde belirli nesneleri tanımlayan gerçek zamanlı bir [nesne algılama algoritmasıdır. YOLO makine öğrenme algoritması, bir görüntüde bulunan nesneleri algılamak için](https://viso.ai/deep-learning/object-detection/)[Derin Evrişimli Sinir Ağı](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/) tarafından öğrenilen özellikleri kullanır . Joseph Redmon ve Ali Farhadi, YOLO sürümleri 1-3'ün yaratıcılarıdır ve YOLO Makine Öğrenmesi (ML) algoritmasının üçüncü sürümü, orijinal ML algoritmasının en doğru sürümüdür.

Joseph Redmon ve Ali Farhadi, YOLO algoritmalarının ilk versiyonunu 2016'da oluşturdular. İkisi daha sonra iki yıl sonra, 2018'de Versiyon 3'ü yayınladılar. YOLOv3, YOLO ve YOLOv2'nin geliştirilmiş bir versiyonudur. YOLO, Keras veya [OpenCV](https://viso.ai/computer-vision/opencv/) derin öğrenme kütüphaneleri kullanılarak uygulanır.

* Evet, kodda derin öğrenme var. LSTM (Long Short-Term Memory) modelini kullanıyor. Şu katmanlardan oluşuyor:

python

Copy

LSTM(150, return\_sequences=True)

Dropout(0.3)

LSTM(100, return\_sequences=True)

Dropout(0.3)

LSTM(100, return\_sequences=True)

Dropout(0.3)

LSTM(50)

Dropout(0.2)

Dense(50, activation='relu')

Dense(25, activation='relu')

Dense(1, activation='linear')

**XGBoost** modeli hala mevcut olduğunu ve import edildi. Dolayısıyla XGBoost, Random Forest ve LightGBM ile birlikte topluluk (ensemble) model yapısının bir parçası olarak kullanılıyor.

Ayrıca, **Monte Carlo simülasyonu** ile ilgili herhangi bir kod veya import ifadesi bulunmuyor, bu yüzden Monte Carlo yöntemi son versiyonda uygulamanın bir parçası değil gibi görünüyor.

Sunum notları:

Uygulamayı yaparken geliştirirken sürekli yeni kütüphaneler geldi ve bazı kütüphaneler python’ın bazı sürümlerine uymuyordu, bu yüzden değiştire değiştire en son python 3.11 de karar kıldım.

Uygulamayı geliştirirken bazı yerlerde ek yüklemeler istedi mesela NVIDIa sitesinden bi şeyler indirdim, sonra visual studio indirme yöneticisinden c++ kodlamalar ile ilgili birşeyler indirmem gerekti sonra vscode da ki pythonda kod çalıştı.

uygulamada şu modeller ve yöntemler mevcut:

1. **LSTM (Long Short-Term Memory)**:
   * Derin öğrenme tabanlı bir yapay sinir ağı modeli.
   * Ana tahmin modeli olarak kullanılıyor ve zaman serisi verilerindeki bağımlılıkları öğrenmede başarılı.
2. **Topluluk (Ensemble) Modelleri**:
   * **Random Forest**: Karar ağaçlarından oluşan bir topluluk modelidir.
   * **XGBoost**: Gradyan artırma yöntemiyle çalışan güçlü bir topluluk modeli.
   * **LightGBM**: Hızlı ve hafif bir gradyan artırma modeli.

**1. LSTM Modeli**

* **Fonksiyon Adı**: build\_lstm\_model
* **Yer**: BitcoinPredictorApp sınıfında, build\_lstm\_model fonksiyonu.
* **Açıklama**: LSTM modeli, Sequential modeli olarak tanımlanıyor ve LSTM katmanları eklenerek yapılandırılıyor. Modelin derin öğrenme tabanlı bir yapay sinir ağı olduğu için zaman serilerindeki uzun vadeli bağımlılıkları öğrenmede kullanılıyor.  
  metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

  Açıklama otomatik olarak oluşturuldu

**2. Topluluk (Ensemble) Modelleri**

* **Fonksiyon Adı**: train\_models ve EnsembleModel sınıfı.
* **Yer**: TrainingThread sınıfında run fonksiyonunda çağrılır ve EnsembleModel sınıfında tanımlanır.
* **Modeller**:
  + **Random Forest**: self.ensemble\_model.rf\_model
  + **XGBoost**: self.ensemble\_model.xgb\_model
  + **LightGBM**: self.ensemble\_model.lgbm\_model
* **Açıklama**: Bu modeller EnsembleModel sınıfında tanımlanmıştır ve train\_models fonksiyonunda eğitilir. Topluluk modeli, LSTM ile birlikte tahmin doğruluğunu artırmak için kullanılır.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yaptığım müdahaleler:

Dip fiyat , zirve fiyatın en fazla beşte biri olabilir. Yani bitcoin düşüş trendine girince en fazla beşte bir fiyatına düşebilir sonra da düşüş durur. Geçmiş fiyat hareketleri buna yaklaşık sonuçlar veriyor.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yükseliş trendleri zamanla gücü azalacak, çünkü piyasa değeri artıyor, dünyada para arzı belli. Bitcoinin sayısı aşırı derece azalmadan fiyatı 100 katrilyon dolar olamaz çünkü dünyadaki para arzı ve doların enflasyonu belli bir oranda ilerliyor. Böyle bir müdahale yaptım.

Accumulation (birikim) aşamasında, belirli bir mini ralli olasılığı tanımlandı. Kodda %20 olasılıkla mini ralliler tetikleniyor ve bu ralliler 2-6 hafta sürebiliyor:

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Birikim aşaması için fiyat aralığı sınırı (%15 fiyat aralığı) ve volatilite sınırı tanımlandı. Böylece bu aşamada fiyat belirli bir aralıkta dalgalanıyor:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ralli aşamalarında ralli gücünün zamanla azaldığı bir yapı oluşturuldu. Logaritmik bir azalma ile belirli bir taban seviyeye kadar düşüyor:

metin, ekran görüntüsü, yazı tipi, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu müdahaleler bitcoin fiyatının gerçeklerden ve realiteden kopmaması için gerçekleşen küçük dokunuşlar oldu.

* Bir **model**, belirli bir algoritmanın verileri analiz edip öğrenmesiyle oluşturulan matematiksel bir yapı veya fonksiyondur. Model, verilerdeki örüntüleri öğrenir ve bu öğrendiği örüntüler sayesinde yeni verilere tahmin veya sınıflandırma yapabilir.

**2. Topluluk (Ensemble) Modelleri ve Çalışma Mantığı**

* Bu projede **Random Forest**, **XGBoost**, ve **LightGBM** modelleri **Ensemble (Topluluk) Modelleri** olarak kullanılıyor. Ensemble, birden fazla modelin çıktısını birleştirerek daha doğru tahminler elde etmeyi amaçlar.
* Bu modeller, verilerdeki karmaşık ilişkileri öğrenerek fiyat tahmininde bulunmak için eğitiliyor.

**3. Modellerin Çalışma Prensibi**

* **Random Forest**, **XGBoost**, ve **LightGBM** gibi algoritmalar **denetimli öğrenme** yöntemleri kullanır. Bu yöntem, modelin geçmiş verilerden öğrenmesi ve gelecekteki sonuçları tahmin etmesi anlamına gelir.
* **Bu modeller nasıl çalışır?**:
  1. **Veri Hazırlığı**: İlk olarak, train\_models gibi bir fonksiyon içinde modelin öğreneceği veriler hazırlanır. Bu verilere, tahmin yapılacak özellikler (örneğin, Bitcoin fiyatı, işlem hacmi, volatilite gibi) ve hedef değerler (örneğin, gelecekteki Bitcoin fiyatı) dahildir.
  2. **Model Eğitimi**: Modeller bu verileri kullanarak eğitilir. Eğitim sırasında model, verilerdeki örüntüleri (örneğin, fiyat ve trend arasındaki ilişki) öğrenir. Her model, verilerdeki farklı yönleri öğrenerek fiyat tahminini daha doğru hale getirir.
  3. **Tahmin Yapma**: Eğitim tamamlandığında, bu modeller yeni veri üzerinde tahmin yapabilir. Örneğin, self.ensemble\_model.rf\_model.predict(yeni\_veri) çağrısı, Random Forest modelini kullanarak yeni veriye göre bir fiyat tahmini yapar.
* self.rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42, n\_jobs=-1)
* Bu satırda, RandomForestRegressor adlı bir model self.rf\_model olarak tanımlanır. Bu model, RandomForestRegressor sınıfını kullanarak bir **Random Forest** algoritması oluşturur.
* **self.ensemble\_model.rf\_model.fit(X, y)** komutu, modeli X ve y verilerini kullanarak eğitir. Burada:
  + X: Özellikler (bağımsız değişkenler) yani modelin tahmin yaparken bakacağı veriler.
  + y: Hedef değerler (bağımlı değişkenler), yani modelin tahmin etmeye çalıştığı asıl değerlerdir.

**5. Yapay Zeka ve Model Tanımlarıyla Ne Elde Ediliyor?**

* **Öğrenme Süreci**: Model, veri örüntülerini ve ilişkileri öğrenir. Örneğin, Random Forest modelinin fit fonksiyonunu çağırdığınızda, model eğitim verilerindeki ilişkilere göre kendini optimize eder.
* **Tahmin Süreci**: Eğitimden sonra, model yeni veriler üzerinde predict fonksiyonu ile tahmin yapabilir. Örneğin, self.ensemble\_model.rf\_model.predict(yeni\_veri) komutuyla Random Forest modeli yeni bir tahmin üretir.

**6. Neden Farklı Modeller Kullanılıyor?**

* **Model Çeşitliliği**: Her modelin öğrenme şekli farklıdır ve verilerin farklı yönlerini yakalayabilir. Örneğin, Random Forest karar ağaçları kullanarak veri içindeki belirgin kalıpları öğrenirken, XGBoost ve LightGBM gradyan artırma yöntemiyle daha ince detayları öğrenir.
* **Ensemble (Topluluk) Yaklaşımı**: Farklı modellerin tahminleri birleştirildiğinde (örneğin, bu projede LSTM ile ensemble modellerin birleşimi), tek bir modelin başarısız olabileceği durumlarda daha sağlam tahminler elde edilebilir.

**LSTM'nin Önemi**

* kodda en öne çıkan yapay zeka modeli **LSTM**. Bunun nedeni, LSTM'nin zaman serisi verilerindeki örüntüleri ve bağımlılıkları öğrenmede çok başarılı olmasıdır. Bitcoin fiyatları gibi zamanla değişen verilere sahip bir durumda, LSTM'nin geçmiş değerleri öğrenerek geleceği tahmin etme yeteneği çok önemlidir.
* **Neden LSTM öncelikli?** LSTM modelini öncelikli olarak kullanmış olmamın sebebi, zaman serisi analizlerinde uzun vadeli bağımlılıkları modellemede güçlü olmasıdır. Diğer modeller, verideki kısa vadeli veya karmaşık ilişkileri öğrenmede etkili olabilir; ancak, asıl tahmin sürecinde uzun vadeli eğilimleri modellemek için LSTM tercih edildi.

**TrainingThread Nedir?**

* **TrainingThread** ifadesi, Türkçe’de "Eğitim İş Parçacığı" anlamına gelir.
* Bu sınıf, modelin arka planda eğitilmesini sağlamak için kullanılır. Kodun çalışması sırasında, kullanıcı arayüzünün donmasını engellemek için eğitim işlemleri farklı bir iş parçacığında (thread) yapılır. Yani, **TrainingThread** sınıfı, model eğitimi sırasında kullanıcı arayüzünün akıcılığını korur ve eğitim işlemini sorunsuzca gerçekleştirir.

**Koddaki Yapay Zeka Modelleri ve İşlevleri**

* **LSTM (Long Short-Term Memory)**:
  + LSTM, zaman serilerindeki uzun vadeli bağımlılıkları öğrenir. Bitcoin fiyatlarının geçmiş hareketlerinden gelecekteki değerleri tahmin etmek için kullanılır.
  + **Uygulamadaki yeri**: build\_lstm\_model fonksiyonunda tanımlanır. Eğitimi ise train\_models fonksiyonunda yapılır.
* **Random Forest**:
  + Random Forest, veriyi birçok karar ağacı üzerinden analiz ederek kısa vadeli bağımlılıkları öğrenir. Bu model, verinin belirli örüntülerini yakalayarak daha istikrarlı tahminler yapabilir.
  + **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

**Random Forest Nedir?**

* **Random Forest**, temel olarak birçok **Karar Ağacı** modelinin bir araya gelerek oluşturduğu bir **topluluk (ensemble) modeli**dir.
* Her bir karar ağacı, veriyi farklı bir açıdan analiz eder ve bağımsız bir tahmin yapar. Random Forest, bu ağaçların tahminlerini birleştirerek daha güçlü ve güvenilir sonuçlar elde eder.
* **“Random” (rastgele)** ismi, her ağacın eğitiminde kullanılan verilerin ve özelliklerin rastgele seçilmesinden gelir.

**Random Forest Nasıl Çalışır?**

Random Forest’ın nasıl çalıştığını basitçe adım adım açıklayalım:

1. **Veri Kümesinin Rastgele Alt Kümelere Bölünmesi**:
   * Random Forest, eğitim verisini birçok küçük alt kümeye böler. Bu alt kümeler, orijinal verinin rastgele seçilmiş örneklerinden oluşur. Bu yöntem **Bootstraping** olarak adlandırılır.
   * Her ağaç farklı bir alt küme üzerinde eğitildiği için, ağaçlar birbirinden bağımsız hale gelir ve fazla öğrenme (overfitting) riski azalır.
2. **Her Ağaç İçin Rastgele Özellik Seçimi**:
   * Her ağacın karar verirken kullanacağı özellikler de rastgele seçilir. Örneğin, bir ağaç fiyat ve işlem hacmi gibi özelliklere bakarken, diğer bir ağaç volatilite ve RSI gibi farklı özelliklere bakabilir.
   * Bu özellik seçim süreci, her ağacın farklı açılardan öğrenmesini sağlar ve modelin çeşitliliğini artırır.
3. **Karar Ağacı Yapısı**:
   * Her bir karar ağacı, veriyi sınıflandırmak veya tahmin etmek için bir dizi "karar" oluşturur. Bu kararlar, **“if-else”** mantığıyla ilerler.
   * Örneğin, bir karar ağacı Bitcoin fiyatının son 7 günlük ortalamasına göre bir tahmin yapabilir: Eğer fiyat ortalaması belirli bir değerden büyükse, fiyatın yükseleceğine dair bir tahminde bulunabilir.
4. **Tahminlerin Birleştirilmesi**:
   * Her bir ağaç bağımsız olarak bir tahmin yapar. Tüm ağaçların yaptığı tahminler toplandıktan sonra bir ortalama alınır (regresyon problemlerinde) veya en çok tekrarlanan tahmin seçilir (sınıflandırma problemlerinde).
   * Bu toplama süreci, modelin daha sağlam olmasını sağlar. Tek bir ağaç hatalı bir tahmin yapsa bile, diğer ağaçların tahminleriyle bu hata dengelenir.

**Neden Random Forest Kullanılır?**

* **Güçlü ve Güvenilir**: Farklı karar ağaçlarının tahminleri bir araya getirildiği için, model daha güvenilir ve sağlam hale gelir. Bu nedenle, Random Forest daha düşük hata oranlarına sahiptir.
* **Aşırı Öğrenme (Overfitting) Riskini Azaltır**: Her ağaç rastgele bir veri alt kümesi üzerinde eğitildiğinden, model veriye fazla uymaz. Bu da modelin yeni veriler üzerinde daha iyi performans göstermesini sağlar.
* Uygulamada, Random Forest modeli EnsembleModel sınıfı içinde self.rf\_model olarak tanımlanmış. train\_models fonksiyonunda, bu model eğitilir:
* self.ensemble\_model.rf\_model.fit(X\_2d, y\_2d)
* **Ne İşe Yarıyor?** Random Forest, uygulamada verinin kısa vadeli bağımlılıklarını öğrenir. Örneğin, son günlerdeki fiyat hareketlerine ve işlem hacmine bakarak gelecekteki fiyat hakkında kısa vadeli tahminlerde bulunabilir.

Özetle:

Random Forest, birçok küçük karar ağacından oluşan bir yapay zeka modelidir. Her bir ağaç kendi başına öğrenir ve tahmin yapar; daha sonra bu tahminler birleştirilerek güçlü bir sonuç elde edilir.

Uygulamanızda, kısa vadeli eğilimleri ve örüntüleri yakalamak için kullanılır ve diğer modellerle birlikte tahmin doğruluğunu artırır.

**Random Forest** ve **LSTM** (Long Short-Term Memory) modelleri, farklı yapay zeka algoritmalarıdır ve farklı görevlerde öne çıkarlar. Bu iki modelin farklarını anlamak için temel özelliklerini ve kullanım alanlarını karşılaştıralım.

**1. Model Türü ve Kullanım Amacı**

* **Random Forest**:
  + **Model Türü**: Karar ağaçlarına dayalı bir **ensemble (topluluk)** yöntemidir.
  + **Kullanım Amacı**: Kısa vadeli bağımlılıkları ve doğrusal olmayan ilişkileri öğrenmekte iyidir. Daha çok **sınıflandırma** ve **regresyon** problemlerinde kullanılır.
  + **Özellikleri**: Verideki örüntüleri ve kısa vadeli ilişkileri yakalayabilir. Zamanla sırayla olan veri bağımlılıklarını dikkate almaz.
* **LSTM**:
  + **Model Türü**: Bir tür **Recurrent Neural Network (RNN)**, yani geri beslemeli sinir ağıdır.
  + **Kullanım Amacı**: **Zaman serisi verileri** gibi ardışık verilerdeki uzun vadeli bağımlılıkları öğrenmekte iyidir. Fiyat tahmini, metin analizi gibi zamanla sırayla gelen verilerde kullanılır.
  + **Özellikleri**: Zaman içinde değişen örüntüleri yakalayabilir. Zaman serisi verilerindeki uzun vadeli bağımlılıkları öğrenebilir, geçmiş veriye dayalı tahminler yapabilir.

**2. Zamanla İlişkili Verilerdeki Performans**

* **Random Forest**:
  + Zamanla ilişkili verilerde doğrudan etkili değildir, çünkü geçmiş verilerle geleceği ilişkilendirmek için özel bir mekanizması yoktur.
  + Her bir veri noktasını bağımsız olarak işler, dolayısıyla veri sıralaması veya zaman bilgisi önemli değildir.
* **LSTM**:
  + Zamanla ilişkili verilerde çok etkilidir, çünkü veri sıralamasını dikkate alarak geçmiş verilere göre geleceği tahmin eder.
  + "Hafıza" hücreleri sayesinde uzun vadeli bağımlılıkları öğrenir ve geçmişteki olayların gelecekteki tahminlerde nasıl bir etkisi olacağını anlamaya çalışır.

**3. Çalışma Prensibi**

* **Random Forest**:
  + Çok sayıda karar ağacı oluşturarak çalışır ve her ağaç, verideki bağımsız örnekleri öğrenir. Her ağacın yaptığı tahminler birleştirilerek nihai sonuç elde edilir.
  + Karar ağaçlarının bağımsızlığı nedeniyle, verinin belirli kısımlarındaki örüntüleri öğrenir, ancak ardışık verileri (örneğin zaman serileri) dikkate almaz.
* **LSTM**:
  + Geri beslemeli sinir ağı olarak çalışır ve verinin zaman sırasına göre işlenmesini sağlar.
  + "Unutma" ve "hatırlama" kapıları sayesinde geçmiş verilerin gelecekte nasıl bir etkisi olacağını anlamaya çalışır. Örneğin, Bitcoin’in fiyatındaki geçmiş dalgalanmalar, LSTM tarafından öğrenilerek gelecekteki tahminlerde kullanılır.

**4. Uygulama Alanları**

* **Random Forest**:
  + Genelde zaman sırasına göre bağımlılık gerektirmeyen veri analizlerinde (örneğin sınıflandırma ve regresyon) kullanılır. Finans, tıp, biyoloji gibi alanlarda, özellikler arasındaki ilişkileri analiz etmek için uygundur.
* **LSTM**:
  + Zaman serisi verilerinde, dil işleme (metin analiz), finansal tahminler, ses tanıma, ve video analizi gibi sıralı verilerle çalışılan durumlarda tercih edilir.

**5. Uygulamadaki Örnek Kullanımlar**

* **Random Forest**: Eğer Bitcoin fiyatının belirli teknik göstergelerle ilişkisini analiz etmek ve bu göstergelerden kısa vadeli fiyat tahmini yapmak istiyorsanız, Random Forest iyi bir seçimdir.
* **LSTM**: Bitcoin fiyatının geçmiş verilerine dayalı olarak gelecekteki fiyatı tahmin etmek istiyorsanız, LSTM daha iyi bir seçenektir. Çünkü fiyat dalgalanmalarındaki zamanla ilişkili örüntüleri öğrenebilir.

**Özet:**

* **Random Forest**: Zaman sırasını dikkate almadan bağımsız örüntüleri öğrenir, kısa vadeli tahminlerde ve sınıflandırma görevlerinde iyidir.
* **LSTM**: Zaman sırasını ve geçmiş bağımlılıkları öğrenir, ardışık verilerdeki örüntüleri yakalayarak uzun vadeli tahminlerde başarılıdır.

**XGBoost**:

* XGBoost, veri içindeki karmaşık ve doğrusal olmayan örüntüleri öğrenmede oldukça etkilidir. Gradyan artırma yöntemiyle çalışarak hata oranını azaltır.
* **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

**XGBoost Nedir?**

* **XGBoost** (Extreme Gradient Boosting), gradyan artırma yöntemiyle çalışan güçlü ve hızlı bir makine öğrenmesi algoritmasıdır.
* Özellikle karmaşık verilerdeki doğrusal olmayan ilişkileri yakalamakta çok başarılıdır. XGBoost, birçok karar ağacını art arda oluşturarak her adımda modeli daha iyi hale getirir.

**Gradyan Artırma Yöntemi Nedir?**

Gradyan artırma yöntemi, bir modeli art arda geliştirerek hataları en aza indirme tekniğidir. Bu yöntem, birden çok zayıf modelin (genellikle karar ağaçları) birleşiminden güçlü bir model oluşturur.

**Gradyan artırmanın nasıl çalıştığını adım adım anlatalım:**

1. **Başlangıç Modeli Oluşturulur**:
   * Gradyan artırma yöntemi, başlangıçta oldukça basit bir tahmin modeliyle başlar. Bu genellikle her veri noktası için ortalama bir değer tahmin eden bir modeldir.
   * İlk model oluşturulduktan sonra, her veri noktası için tahminler yapılır ve hatalar (gerçek değer ile tahmin arasındaki fark) hesaplanır.
2. **Hataların Öğrenilmesi**:
   * İlk modelin yaptığı hatalar analiz edilir. Gradyan artırma yöntemi, bu hataları en aza indirmek için çalışır.
   * Hataları anlamak için **gradyan** (eğim) hesaplanır. Bu eğim, modelin her veri noktası için ne kadar yanlış tahmin yaptığını gösterir. Yani, her bir veri noktası için "Bu tahmin ne kadar yanlış?" sorusunun cevabını verir.
3. **Yeni Ağaç Eklenmesi**:
   * Gradyan artırma yöntemi, hataları düzeltecek şekilde yeni bir karar ağacı ekler. Bu yeni ağaç, bir önceki modelin yaptığı hataları "öğrenmeye" çalışır.
   * Örneğin, ilk model bazı veri noktalarında çok yüksek, bazı veri noktalarında çok düşük tahminler yapmışsa, eklenen yeni ağaç bu hataları dengelemeye çalışır.
4. **Ağaçların Birleştirilmesi**:
   * Eklenen her yeni ağaç, bir önceki modelin hatalarını düzeltmeye yönelik olduğundan, zamanla model daha doğru tahminler yapmaya başlar.
   * Gradyan artırma süreci, her bir ağacın katkısını küçük adımlarla ekleyerek hataları kademeli olarak azaltır. Böylece model, her yeni eklenen ağaçla birlikte daha hassas hale gelir.
5. **Tüm Ağaçların Toplanması**:
   * Gradyan artırma, belirli sayıda ağaç ekleyene veya hatalar çok küçük hale gelene kadar devam eder.
   * Sonuç olarak, tüm ağaçların tahminleri birleştirilir ve daha güçlü, daha doğru bir model elde edilir.

**XGBoost ve Gradyan Artırma Yöntemi Arasındaki İlişki**

* **XGBoost**, gradyan artırma yönteminin gelişmiş ve optimize edilmiş bir versiyonudur. Standart gradyan artırma algoritmalarına göre daha hızlı ve verimli çalışır.
* XGBoost, **daha iyi genel performans**, **hızlı eğitim süresi**, ve **daha düşük hata oranı** sağlamak için optimizasyon teknikleri (örneğin, daha hızlı hesaplama için paralel işleme) kullanır.

**Neden Gradyan Artırma ve XGBoost Kullanıyoruz?**

* **Doğrusal Olmayan Örüntüleri Yakalar**: XGBoost, verideki karmaşık ilişkileri öğrenme konusunda çok iyidir. Bitcoin fiyat tahmini gibi doğrusal olmayan yapıya sahip verilerde iyi sonuç verir.
* **Düşük Hata Oranı Sağlar**: Her adımda hataları azaltarak daha doğru tahminler yapar. Bu yüzden finansal verilerde çok kullanılır.

**Uygulamadaki Yeri ve Kullanımı**

* XGBoost, EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda fit fonksiyonu ile eğitilir:
* self.ensemble\_model.xgb\_model.fit(X\_2d, y\_2d)

Bu kod, modelin verilerdeki örüntüleri öğrenmesini ve tahminler yapmasını sağlar. Yani XGBoost, veriyi analiz ederek kısa vadeli tahminler yapar ve tahmin doğruluğunu artırır.

**LightGBM**:

* LightGBM, hızlı bir gradyan artırma modelidir ve büyük veri kümelerinde bile yüksek hızda çalışabilir. Verideki detayları hızlı bir şekilde öğrenir.
* **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

17 kasım 2024

Uygulama tahmin yapıyor fakat tahmin et’e tekrar basınca farklı bir tahmin yapıyor. Bu program hakkında neyi gösteriyor? Programın tahminleri iddialı değil mi? Akademik tezleri incelerken herkes programının başarı oranından bahsediyordu. Benim uygulamam her seferinde farklı bir tahmin yapıyor. Peki böyle bir durumda bu uygulamanın bir başarı oranı şudur budur diyebilir miyiz? Uygulamayı tek bir tahmin yapar hale mi dönüştürelim yoksa böyle kalsın mı sence?

Def show

Bu önemli bir gözlem ve soru. Programınızın davranışını ve yapısını analiz edeyim:

1. **Rastgelsalık ve Stokastik Yapı:** Program içinde birçok rastgele (random) eleman kullanılıyor. Örneğin:

* np.random.uniform
* np.random.normal
* np.random.randint gibi fonksiyonlar kullanılıyor. Bu, her çalıştırmada farklı tahminler üretilmesine neden oluyor.