Contents

[Örnek 1 – BASİT ÖRNEK 1](#_Toc180341304)

[ÖRNEK 2 – DAHA ÇOK ÖZELLİK İLE 6](#_Toc180341305)

[ÖRNEK 3 - XGBOOST 11](#_Toc180341306)

[ÖRNEK – 4 Deep Learning 16](#_Toc180341307)

[ÖRNEK 5 - LSTM 22](#_Toc180341308)

# Örnek 1 – BASİT ÖRNEK

Bu projede, belirtilen hisseler için 2024 Eylül günleri itibariyle "al", "sat" veya "tut" kararlarını tahmin eden bir model geliştireceğiz. Kullanacağımız veri, `yfinance` kütüphanesi aracılığıyla Yahoo Finance API’den çekilecek ve Python'da farklı analiz ve modelleme teknikleriyle işlenecektir. Modeli geliştirmek için makine öğrenmesi tekniklerinden biri olan sınıflandırma modellerini kullanacağız.

Adımlarımız:

1. \*\*Veri çekme\*\*: `yfinance` kullanarak belirtilen hisselerin geçmiş verilerini çekeceğiz.

2. \*\*Veri ön işleme\*\*: Veriyi hazırlayıp özellik mühendisliği ile yeni değişkenler yaratacağız.

3. \*\*Modelleme\*\*: Sınıflandırma modeli kurarak "al", "sat" veya "tut" kararlarını tahmin edeceğiz.

4. \*\*Model değerlendirmesi\*\*: Modelin performansını değerlendireceğiz.

### 1. Veri Çekme

İlk olarak, hisselerin geçmiş verilerini çekeceğiz. 2024 Eylül ayına kadar olan verileri alacağız.

#### Python Kodları

```python

import yfinance as yf

import pandas as pd

# Hisse sembollerini belirleyelim

symbols = ['AAPL', 'NVDA', 'MSFT', 'AMZN', 'META', 'ADBE', 'TSLA', 'FFIE', 'ASTI', 'ALLR']

# Her bir hisse için verileri indirme

data = yf.download(symbols, start='2020-01-01', end='2024-09-30')

# İlk birkaç satırı görelim

print(data.head())

```

Bu kod ile belirtilen hisseler için 2020-2024 yılları arasındaki verileri indireceğiz.

### 2. Veri Ön İşleme ve Özellik Mühendisliği

Veriyi işleyip, modele uygun hale getirmemiz gerekiyor. Modelleme için fiyat değişim oranları, hareketli ortalamalar gibi özellikler yaratacağız. Bu özellikler, hisse senedi fiyatındaki değişimleri modellemeye yardımcı olacaktır.

#### Python Kodları

```python

# Veri ön işleme ve özellik mühendisliği

# Hisse fiyatı değişim oranları ve hareketli ortalamalar gibi teknik göstergeleri hesaplayalım

# Yüzde değişim

data['Pct\_Change'] = data['Adj Close'].pct\_change()

# 7 günlük hareketli ortalama

data['MA7'] = data['Adj Close'].rolling(window=7).mean()

# 30 günlük hareketli ortalama

data['MA30'] = data['Adj Close'].rolling(window=30).mean()

# Relative Strength Index (RSI)

def rsi(df, n=14):

delta = df['Adj Close'].diff()

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

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

rs = gain / loss

return 100 - (100 / (1 + rs))

data['RSI'] = rsi(data)

# Boş değerleri kaldıralım

data = data.dropna()

# İlk birkaç satırı görelim

print(data.head())

```

Bu kod, yüzde değişim, hareketli ortalamalar ve RSI gibi teknik göstergeleri hesaplayacaktır.

### 3. Hedef Değişkenin Tanımlanması

Modelimizin hedefi, hisseler için "al", "sat" veya "tut" kararlarını tahmin etmek olacak. Bunu, belirli bir eşik değerine göre karar vererek yapacağız. Örneğin, eğer hisse fiyatı belirli bir yüzde artarsa "al", düşerse "sat" olarak sınıflandırabiliriz.

#### Python Kodları

```python

# Hedef değişken oluşturma

def get\_signal(df):

# Eğer yüzde değişim > 0.02 -> 'Buy', yüzde değişim < -0.02 -> 'Sell', aksi halde 'Hold'

conditions = [

(df['Pct\_Change'] > 0.02),

(df['Pct\_Change'] < -0.02)

]

choices = ['Buy', 'Sell']

df['Signal'] = np.select(conditions, choices, default='Hold')

return df

data = get\_signal(data)

# İlk birkaç satırı inceleyelim

print(data[['Adj Close', 'Pct\_Change', 'Signal']].tail())

```

Bu fonksiyon, yüzde değişim eşiklerine göre "Buy", "Sell" ve "Hold" sinyalleri oluşturacaktır.

### 4. Modelleme

Veri hazırlandıktan sonra, sınıflandırma modelleriyle tahminler yapabiliriz. Random Forest gibi bir model kullanarak "al", "sat" veya "tut" kararlarını tahmin edeceğiz.

#### Python Kodları

```python

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Model için girdi ve hedef değişkenleri ayıralım

X = data[['Pct\_Change', 'MA7', 'MA30', 'RSI']]

y = data['Signal']

# Veriyi eğitim ve test setine bölelim

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Random Forest modeli

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# Tahminler

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

# Model değerlendirmesi

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

```

Bu kod, veriyi eğitim ve test setlerine böler, ardından Random Forest modelini eğitir ve test seti üzerinde tahminler yapar.

### 5. Model Değerlendirmesi

Model performansını değerlendirmek için doğruluk (accuracy), precision, recall ve F1 score gibi metriklere bakacağız. Bu metrikler, modelin "al", "sat" ve "tut" kararlarını ne kadar iyi tahmin ettiğini gösterecektir.

#### Python Kodları

```python

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Hold', 'Buy', 'Sell'], yticklabels=['Hold', 'Buy', 'Sell'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

```

Bu kod, tahmin edilen sinyaller ile gerçek sinyaller arasındaki uyumu gösteren bir confusion matrix oluşturur.

### Sonuç

Bu analiz ve modelleme ile, 2024 Eylül ayında bu 10 hisse için al, sat veya tut kararlarını tahmin edebiliriz. Modelin performansını değerlendirdikten sonra, gerekirse farklı modeller deneyebilir veya parametre ayarlamaları yapabilirsiniz.

# ÖRNEK 2 – DAHA ÇOK ÖZELLİK İLE

Daha hassas bir model geliştirmek için, mevcut veri setimize daha fazla özellik ekleyebiliriz. Özellik mühendisliği, bir modelin performansını önemli ölçüde artırabilecek güçlü bir tekniktir. Bu adımda, daha fazla teknik gösterge ve zaman serisi özellikleri oluşturarak modelimizi geliştireceğiz. Yeni özellikler, modelin daha fazla bilgiye erişmesini sağlar ve bu da daha doğru tahminler yapmasına yardımcı olabilir.

### Yeni Özellikler

1. \*\*Momentum\*\*: Bir hissenin fiyatının kısa vadede yukarı ya da aşağı hareketini gösterir.

2. \*\*Bollinger Bands\*\*: Hisse senedinin volatilitesini ve aşırı alım/aşırı satım durumlarını tespit etmeye yarar.

3. \*\*MACD (Moving Average Convergence Divergence)\*\*: Fiyat trendlerinin gücünü ve yönünü belirlemeye yardımcı olur.

4. \*\*Volatility\*\*: Hisse senedinin ne kadar dalgalandığını ölçer.

5. \*\*On-Balance Volume (OBV)\*\*: Hacim akışı ile fiyat hareketlerini birleştirir.

6. \*\*Williams %R\*\*: Aşırı alım veya aşırı satım durumlarını tespit eder.

### 1. Veri Çekme

Önceden kullanılan veri çekme kodu ile aynı şekilde hisseler için veri çekeceğiz:

```python

import yfinance as yf

import pandas as pd

# Hisse sembollerini belirleyelim

symbols = ['AAPL', 'NVDA', 'MSFT', 'AMZN', 'META', 'ADBE', 'TSLA', 'FFIE', 'ASTI', 'ALLR']

# Verileri indirme

data = yf.download(symbols, start='2020-01-01', end='2024-09-30')

data = data.dropna()

import os

import yfinance as yf

import pandas as pd

import pprint

# Set the proxy

os.environ['http\_proxy'] = 'http://tekprxv2:80'

os.environ['https\_proxy'] = 'http://tekprxv2:80'

# Define the ticker symbol

symbol = 'AAPL' # Apple Inc.

# Get data on this ticker

inst = yf.Ticker(symbol)

df1 = inst.quarterly\_cash\_flow

df2 = inst.quarterly\_incomestmt

df3 = inst.quarterly\_balance\_sheet

df4 = inst.quarterly\_financials

df5 = inst.history(period='10y')

```

### 2. Yeni Özelliklerin Hesaplanması

#### 2.1 Momentum

```python

# Momentum hesaplama (10 gün)

data['Momentum'] = data['Adj Close'].diff(10)

```

#### 2.2 Bollinger Bands

Bollinger Bantları, fiyatın hareketli ortalamasını ve belirli bir standart sapma aralığındaki fiyatları gösterir:

```python

# Bollinger Band hesaplama (20 günlük)

data['MA20'] = data['Adj Close'].rolling(window=20).mean()

data['BB\_Upper'] = data['MA20'] + 2\*data['Adj Close'].rolling(window=20).std()

data['BB\_Lower'] = data['MA20'] - 2\*data['Adj Close'].rolling(window=20).std()

```

#### 2.3 MACD (Moving Average Convergence Divergence)

MACD, kısa ve uzun vadeli hareketli ortalamalar arasındaki farkı gösterir.

```python

# MACD hesaplama

data['EMA12'] = data['Adj Close'].ewm(span=12, adjust=False).mean()

data['EMA26'] = data['Adj Close'].ewm(span=26, adjust=False).mean()

data['MACD'] = data['EMA12'] - data['EMA26']

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

```

#### 2.4 Volatility

Volatilite, standart sapma kullanılarak hesaplanır:

```python

# Volatilite hesaplama (10 günlük)

data['Volatility'] = data['Adj Close'].rolling(window=10).std()

```

#### 2.5 On-Balance Volume (OBV)

Hacim ve fiyat hareketlerine dayalı olarak hesaplanır.

```python

# OBV hesaplama

data['OBV'] = (data['Volume'] \* ((data['Adj Close'] - data['Adj Close'].shift(1)) > 0).astype(int) \* 2 - 1).cumsum()

```

#### 2.6 Williams %R

Williams %R, aşırı alım ve aşırı satım durumlarını tespit eder.

```python

# Williams %R hesaplama (14 günlük)

data['Williams\_%R'] = (data['High'].rolling(14).max() - data['Adj Close']) / (data['High'].rolling(14).max() - data['Low'].rolling(14).min()) \* -100

```

### 3. Modelleme

Yeni özellikler eklediğimiz veri setini modelleme için hazırlayalım ve Random Forest gibi bir sınıflandırma algoritmasını kullanalım.

```python

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, accuracy\_score

# Yeni özellikleri kullanarak veriyi ve hedefi ayırma

X = data[['Pct\_Change', 'MA7', 'MA30', 'RSI', 'Momentum', 'BB\_Upper', 'BB\_Lower', 'MACD', 'Signal\_Line', 'Volatility', 'OBV', 'Williams\_%R']]

y = data['Signal'] # Önceden oluşturulmuş "Buy", "Sell", "Hold" hedef değişkeni

# Veriyi eğitim ve test setlerine bölme

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Random Forest modelini oluşturma ve eğitme

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# Test verisi üzerinde tahminler yapma

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

# Model performansını değerlendirme

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

```

### 4. Model Değerlendirme

Modelin performansını daha iyi anlamak için doğruluk (accuracy), precision, recall ve F1-score gibi metrikler kullanacağız.

#### Modelin Performansı İçin Confusion Matrix

```python

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Hold', 'Buy', 'Sell'], yticklabels=['Hold', 'Buy', 'Sell'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

```

### 5. Sonuç

Eklediğimiz yeni özelliklerle (Momentum, Bollinger Bands, MACD, Volatility, OBV ve Williams %R) daha kapsamlı bir model geliştirdik. Bu model, hisse senedi fiyat hareketlerini daha iyi anlamamıza yardımcı olacak ve "al", "sat" veya "tut" kararlarını daha hassas bir şekilde tahmin edecektir. Modelin performansına göre, diğer algoritmalarla da (XGBoost, SVM gibi) testler yaparak daha iyi sonuçlar elde edebilirsiniz.

Geliştirilen bu özellik mühendisliği ile daha karmaşık ve hassas bir tahmin modeli ortaya çıkacaktır.

# ÖRNEK 3 - XGBOOST

XGBoost, sınıflandırma ve regresyon problemleri için oldukça popüler ve güçlü bir algoritmadır. Daha önce geliştirdiğimiz özellikleri kullanarak XGBoost ile modelleme yapabiliriz. Bu adımda, XGBoost kullanarak hisseler için "al", "sat" ve "tut" sinyallerini tahmin edeceğiz.

Aşağıdaki adımlarla XGBoost modelini oluşturacağız:

1. \*\*Veri Hazırlığı\*\*: Veri setini XGBoost için uygun hale getirme.

2. \*\*Model Eğitimi\*\*: XGBoost modeli ile veriyi eğitme.

3. \*\*Tahmin ve Değerlendirme\*\*: Modelin performansını değerlendirme.

### 1. Veri Hazırlığı

Önce veri setimizi modelleme için uygun hale getirelim.

#### Python Kodları

```python

import yfinance as yf

import pandas as pd

import numpy as np

# Hisse sembollerini belirleyelim

symbols = ['AAPL', 'NVDA', 'MSFT', 'AMZN', 'META', 'ADBE', 'TSLA', 'FFIE', 'ASTI', 'ALLR']

# Verileri indirme

data = yf.download(symbols, start='2020-01-01', end='2024-09-30')

data = data.dropna()

# Yeni özellikler ekleyelim (özellik mühendisliği)

data['Pct\_Change'] = data['Adj Close'].pct\_change()

data['MA7'] = data['Adj Close'].rolling(window=7).mean()

data['MA30'] = data['Adj Close'].rolling(window=30).mean()

data['Momentum'] = data['Adj Close'].diff(10)

data['MA20'] = data['Adj Close'].rolling(window=20).mean()

data['BB\_Upper'] = data['MA20'] + 2\*data['Adj Close'].rolling(window=20).std()

data['BB\_Lower'] = data['MA20'] - 2\*data['Adj Close'].rolling(window=20).std()

data['EMA12'] = data['Adj Close'].ewm(span=12, adjust=False).mean()

data['EMA26'] = data['Adj Close'].ewm(span=26, adjust=False).mean()

data['MACD'] = data['EMA12'] - data['EMA26']

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

data['Volatility'] = data['Adj Close'].rolling(window=10).std()

data['OBV'] = (data['Volume'] \* ((data['Adj Close'] - data['Adj Close'].shift(1)) > 0).astype(int) \* 2 - 1).cumsum()

data['Williams\_%R'] = (data['High'].rolling(14).max() - data['Adj Close']) / (data['High'].rolling(14).max() - data['Low'].rolling(14).min()) \* -100

# Boş verileri kaldır

data = data.dropna()

# Hedef değişkeni oluştur (Buy, Sell, Hold)

def get\_signal(df):

conditions = [

(df['Pct\_Change'] > 0.02),

(df['Pct\_Change'] < -0.02)

]

choices = ['Buy', 'Sell']

df['Signal'] = np.select(conditions, choices, default='Hold')

return df

data = get\_signal(data)

# Girdi ve hedef değişkenleri ayıralım

X = data[['Pct\_Change', 'MA7', 'MA30', 'Momentum', 'BB\_Upper', 'BB\_Lower', 'MACD', 'Signal\_Line', 'Volatility', 'OBV', 'Williams\_%R']]

y = data['Signal']

```

### 2. XGBoost ile Model Eğitimi

XGBoost modelini kullanmak için önce `xgboost` kütüphanesini yüklemeliyiz:

```bash

pip install xgboost

```

#### Python Kodları

```python

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

import xgboost as xgb

from sklearn.metrics import accuracy\_score, classification\_report

# Veriyi eğitim ve test setine bölelim

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# XGBoost sınıflandırıcı modelini oluşturalım

model = xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss')

# Modeli eğitelim

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

# Test seti üzerinde tahmin yapalım

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

# Modelin doğruluk ve performansını inceleyelim

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print(classification\_report(y\_test, y\_pred))

```

### 3. Performans Değerlendirmesi

Modelin performansını değerlendirmek için doğruluk (accuracy), precision, recall ve F1-score gibi metrikleri kullanacağız.

#### Python Kodları

```python

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Hold', 'Buy', 'Sell'], yticklabels=['Hold', 'Buy', 'Sell'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

```

Bu grafik, tahmin edilen sinyaller ile gerçek sinyaller arasındaki uyumu görsel olarak incelememizi sağlar.

### 4. Model Optimizasyonu (Hiperparametre Ayarı)

Modelin performansını daha da iyileştirmek için XGBoost'un hiperparametrelerini optimize edebiliriz. Örneğin, `n\_estimators`, `max\_depth`, `learning\_rate` gibi parametreler üzerinde ayarlamalar yaparak modeli daha hassas hale getirebiliriz.

#### Python Kodları

```python

# Hiperparametre ayarı ile model eğitimi

model = xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss', n\_estimators=200, max\_depth=5, learning\_rate=0.1)

# Modeli tekrar eğit ve değerlendir

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

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

# Yeni sonuçları değerlendir

accuracy = accuracy\_score(y\_test, y\_pred)

print("Improved Accuracy:", accuracy)

print(classification\_report(y\_test, y\_pred))

```

Bu ayarlamalarla, modelin performansını daha iyi hale getirmeye çalışabiliriz.

### Sonuç

Bu adımlarla XGBoost kullanarak daha hassas bir model oluşturduk. Model, hisse senedi fiyatlarındaki değişimlere göre "al", "sat" veya "tut" sinyallerini tahmin edebiliyor. Ayrıca, daha fazla özellik ekleyerek ve hiperparametre ayarları yaparak modelin doğruluğunu artırmayı başardık. Modelin performansı, veri setine ve seçilen hiperparametrelere göre değişebilir, bu nedenle farklı parametre ayarlarını denemek daha iyi sonuçlar verebilir.

# ÖRNEK – 4 Deep Learning

Deep learning modelleri, özellikle büyük ve karmaşık veri setlerinde, sınıflandırma ve tahmin problemleri için güçlü sonuçlar verebilir. Hisse senedi fiyatlarına dayalı "al", "sat" ve "tut" sinyallerini tahmin etmek için bir \*\*derin öğrenme\*\* modeli kullanabiliriz. Bu tür problemler için yaygın olarak \*\*Yapay Sinir Ağları (Artificial Neural Networks - ANN)\*\* veya daha gelişmiş modeller (LSTM gibi zaman serisi verileri için uygun modeller) kullanılabilir. Biz, temel bir \*\*Yapay Sinir Ağı\*\* modeli ile başlayacağız ve modelin performansını inceleyeceğiz.

### 1. Keras ve TensorFlow Kurulumu

Derin öğrenme modelini eğitmek için `Keras` ve `TensorFlow` kütüphanelerini kullanacağız. TensorFlow'u yüklemek için şu komutu çalıştırabilirsiniz:

```bash

pip install tensorflow

```

### 2. Yapay Sinir Ağı (ANN) ile Model Eğitimi

Aşağıda, temel bir Yapay Sinir Ağı (ANN) modeli kurup eğiterek hisse senedi sinyallerini tahmin etmeye çalışacağız.

#### Python Kodları

```python

import yfinance as yf

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

# Hisse sembollerini belirleyelim

symbols = ['AAPL', 'NVDA', 'MSFT', 'AMZN', 'META', 'ADBE', 'TSLA', 'FFIE', 'ASTI', 'ALLR']

# Verileri indirme

data = yf.download(symbols, start='2020-01-01', end='2024-09-30')

data = data.dropna()

# Özellik mühendisliği (önceki adımlardaki gibi)

data['Pct\_Change'] = data['Adj Close'].pct\_change()

data['MA7'] = data['Adj Close'].rolling(window=7).mean()

data['MA30'] = data['Adj Close'].rolling(window=30).mean()

data['Momentum'] = data['Adj Close'].diff(10)

data['MA20'] = data['Adj Close'].rolling(window=20).mean()

data['BB\_Upper'] = data['MA20'] + 2\*data['Adj Close'].rolling(window=20).std()

data['BB\_Lower'] = data['MA20'] - 2\*data['Adj Close'].rolling(window=20).std()

data['EMA12'] = data['Adj Close'].ewm(span=12, adjust=False).mean()

data['EMA26'] = data['Adj Close'].ewm(span=26, adjust=False).mean()

data['MACD'] = data['EMA12'] - data['EMA26']

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

data['Volatility'] = data['Adj Close'].rolling(window=10).std()

data['OBV'] = (data['Volume'] \* ((data['Adj Close'] - data['Adj Close'].shift(1)) > 0).astype(int) \* 2 - 1).cumsum()

data['Williams\_%R'] = (data['High'].rolling(14).max() - data['Adj Close']) / (data['High'].rolling(14).max() - data['Low'].rolling(14).min()) \* -100

# Boş verileri kaldır

data = data.dropna()

# Hedef değişkeni oluştur (Buy, Sell, Hold)

def get\_signal(df):

conditions = [

(df['Pct\_Change'] > 0.02),

(df['Pct\_Change'] < -0.02)

]

choices = ['Buy', 'Sell']

df['Signal'] = np.select(conditions, choices, default='Hold')

return df

data = get\_signal(data)

# Girdi ve hedef değişkenlerini ayıralım

X = data[['Pct\_Change', 'MA7', 'MA30', 'Momentum', 'BB\_Upper', 'BB\_Lower', 'MACD', 'Signal\_Line', 'Volatility', 'OBV', 'Williams\_%R']]

y = data['Signal']

# Label Encoding (y'yi sayısal hale getirmek için)

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Veriyi normalleştirme

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Veriyi eğitim ve test setlerine bölelim

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Yapay Sinir Ağı modeli oluşturma

model = Sequential()

# Girdi katmanı ve ilk gizli katman

model.add(Dense(128, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dropout(0.3))

# İkinci gizli katman

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.3))

# Çıkış katmanı

model.add(Dense(3, activation='softmax')) # 3 sınıf (Buy, Sell, Hold)

# Modeli derleme

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Modeli eğitme

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Performansı değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy}')

```

### 3. Performans Değerlendirmesi

Modelin performansını eğitim ve doğrulama seti üzerinde nasıl sonuç verdiğini grafiklerle inceleyebiliriz.

#### Python Kodları

```python

import matplotlib.pyplot as plt

# Eğitim ve doğrulama doğruluğu grafiği

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(loc='upper left')

plt.show()

# Eğitim ve doğrulama kaybı grafiği

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(loc='upper right')

plt.show()

```

### 4. Modelin Performansını Test Etme

Modelin performansını test verisi üzerinde değerlendirdikten sonra, tahminler yaparak "Buy", "Sell" veya "Hold" sinyallerini nasıl tahmin ettiğine bakabiliriz.

#### Python Kodları

```python

# Test verisi üzerindeki tahminler

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

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Gerçek ve tahmin edilen sınıfları karşılaştıralım

from sklearn.metrics import classification\_report, confusion\_matrix

print(classification\_report(y\_test, y\_pred\_classes, target\_names=label\_encoder.classes\_))

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred\_classes)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

```

### Sonuç

Bu derin öğrenme modeli ile, hisse senedi fiyatlarına dayalı olarak "al", "sat" ve "tut" sinyallerini tahmin etmeye çalıştık. Yapay Sinir Ağı (ANN) modeli, verileri öğrenerek tahminlerde bulundu ve modelin doğruluğunu artırmak için dropout katmanları ekleyerek aşırı öğrenmeyi önledik. Eğitim sırasında model performansı grafiklerle takip edilerek doğrulama seti üzerindeki performans da kontrol edildi.

Bu model daha karmaşık hale getirilebilir ve optimizasyon teknikleri (örneğin, hiperparametre ayarı, daha fazla veri kullanımı) ile performansı artırılabilir. Ayrıca, bu model zaman serisi verilerini daha iyi modellemek için LSTM gibi daha gelişmiş derin öğrenme yöntemlerine de genişletilebilir.

# ÖRNEK 5 - LSTM

LSTM (Long Short-Term Memory), zaman serisi verileriyle çalışırken güçlü sonuçlar veren bir tür RNN (Recurrent Neural Network) modelidir. Hisse senedi fiyatları gibi zaman serisi verilerinde geçmiş fiyatların gelecekteki fiyatları tahmin etmede önemli olduğu düşünüldüğünde, LSTM modelleri oldukça etkili olabilir.

Bu adımda, LSTM modeli kullanarak "al", "sat" ve "tut" sinyallerini tahmin etmeye çalışacağız.

### 1. Veri Hazırlığı

LSTM modeli için veriyi yeniden şekillendirmemiz gerekecek, çünkü LSTM zaman serisi verileriyle çalışır ve model girdisi olarak diziler bekler. Bu nedenle, veri setimizi LSTM'nin anlayabileceği bir formata dönüştüreceğiz.

#### Python Kodları

```python

import yfinance as yf

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from tensorflow.keras.models import Sequential

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

# Hisse sembollerini belirleyelim

symbols = ['AAPL', 'NVDA', 'MSFT', 'AMZN', 'META', 'ADBE', 'TSLA', 'FFIE', 'ASTI', 'ALLR']

# Verileri indirme

data = yf.download(symbols, start='2020-01-01', end='2024-09-30')

data = data.dropna()

# Özellik mühendisliği (önceki adımlardaki gibi)

data['Pct\_Change'] = data['Adj Close'].pct\_change()

data['MA7'] = data['Adj Close'].rolling(window=7).mean()

data['MA30'] = data['Adj Close'].rolling(window=30).mean()

data['Momentum'] = data['Adj Close'].diff(10)

data['MA20'] = data['Adj Close'].rolling(window=20).mean()

data['BB\_Upper'] = data['MA20'] + 2\*data['Adj Close'].rolling(window=20).std()

data['BB\_Lower'] = data['MA20'] - 2\*data['Adj Close'].rolling(window=20).std()

data['EMA12'] = data['Adj Close'].ewm(span=12, adjust=False).mean()

data['EMA26'] = data['Adj Close'].ewm(span=26, adjust=False).mean()

data['MACD'] = data['EMA12'] - data['EMA26']

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

data['Volatility'] = data['Adj Close'].rolling(window=10).std()

data['OBV'] = (data['Volume'] \* ((data['Adj Close'] - data['Adj Close'].shift(1)) > 0).astype(int) \* 2 - 1).cumsum()

data['Williams\_%R'] = (data['High'].rolling(14).max() - data['Adj Close']) / (data['High'].rolling(14).max() - data['Low'].rolling(14).min()) \* -100

# Boş verileri kaldır

data = data.dropna()

# Hedef değişkeni oluştur (Buy, Sell, Hold)

def get\_signal(df):

conditions = [

(df['Pct\_Change'] > 0.02),

(df['Pct\_Change'] < -0.02)

]

choices = ['Buy', 'Sell']

df['Signal'] = np.select(conditions, choices, default='Hold')

return df

data = get\_signal(data)

# Girdi ve hedef değişkenlerini ayıralım

X = data[['Pct\_Change', 'MA7', 'MA30', 'Momentum', 'BB\_Upper', 'BB\_Lower', 'MACD', 'Signal\_Line', 'Volatility', 'OBV', 'Williams\_%R']]

y = data['Signal']

# Label Encoding (y'yi sayısal hale getirmek için)

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Veriyi normalleştirme

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# LSTM modeline uygun hale getirmek için veriyi yeniden şekillendirelim

def reshape\_for\_lstm(X, y, time\_steps=30):

X\_lstm, y\_lstm = [], []

for i in range(len(X) - time\_steps):

X\_lstm.append(X[i:i + time\_steps, :])

y\_lstm.append(y[i + time\_steps])

return np.array(X\_lstm), np.array(y\_lstm)

# 30 günlük zaman penceresi ile veriyi LSTM'e uygun hale getirelim

X\_lstm, y\_lstm = reshape\_for\_lstm(X, y)

# Veriyi eğitim ve test setlerine ayıralım

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_lstm, y\_lstm, test\_size=0.3, random\_state=42)

```

### 2. LSTM Modeli Kurma

LSTM modelini oluşturup eğiteceğiz. Modelimizde birkaç katman olacak: LSTM katmanı, Dropout katmanı (overfitting'i önlemek için), ve bir Dense katmanı çıkış katmanı olarak kullanacağız.

#### Python Kodları

```python

# LSTM modelini oluşturalım

model = Sequential()

# İlk LSTM katmanı

model.add(LSTM(128, input\_shape=(X\_train.shape[1], X\_train.shape[2]), return\_sequences=True))

model.add(Dropout(0.3))

# İkinci LSTM katmanı

model.add(LSTM(64, return\_sequences=False))

model.add(Dropout(0.3))

# Çıkış katmanı

model.add(Dense(3, activation='softmax')) # 3 sınıf (Buy, Sell, Hold)

# Modeli derleme

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Modeli eğitme

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test))

# Performansı değerlendirme

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy}')

```

### 3. Performans Değerlendirmesi

Modelin eğitim ve doğrulama seti üzerindeki performansını grafiklerle inceleyebiliriz.

#### Python Kodları

```python

import matplotlib.pyplot as plt

# Eğitim ve doğrulama doğruluğu grafiği

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(loc='upper left')

plt.show()

# Eğitim ve doğrulama kaybı grafiği

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(loc='upper right')

plt.show()

```

### 4. Modelin Performansını Test Etme

LSTM modelinin performansını test verisi üzerinde değerlendirdikten sonra, tahminler yaparak "Buy", "Sell" veya "Hold" sinyallerini nasıl tahmin ettiğine bakabiliriz.

#### Python Kodları

```python

# Test verisi üzerindeki tahminler

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

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Gerçek ve tahmin edilen sınıfları karşılaştıralım

from sklearn.metrics import classification\_report, confusion\_matrix

print(classification\_report(y\_test, y\_pred\_classes, target\_names=label\_encoder.classes\_))

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred\_classes)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

```

### Sonuç

LSTM modeli, zaman serisi verileriyle daha iyi performans gösterebilecek bir modeldir. Bu model, belirli bir zaman aralığına dayanarak hisse senedi fiyat hareketlerini inceleyip "al", "sat" ve "tut" sinyallerini tahmin edebilir. Modelin doğruluğunu artırmak için, hiperparametre ayarlarını daha da iyileştirebilir veya daha büyük bir veri seti kullanarak modelin performansını geliştirebilirsiniz.

LSTM gibi zaman serisi odaklı modeller, hisse senedi fiyatlarını anlamada ve gelecekteki fiyat hareketlerini tahmin etmede güçlü araçlar sunar.