PROGNOZOWANIE CEN AKCJI - RANDOM FOREST W PYTHONIE

Maciej Otto
Laura Milczanowska

WPROWADZENIE

CEL: Stworzenie aplikacji webowej do prognozowania wybranych przez użytkownika kursów aktywów finansowych oraz generowania sygnałów dla tych aktywów:

- sygnał wzrostowy
- sygnał spadkowy
- brak sygnału / neutralny sygnał

Funkcjonalność aplikacji

01

Pobieranie danych historycznych (Stooq)

02

Przetwarzanie szeregów czasowych na problem uczenia nadzorowanego (Lags)

03

Trenowanie modelu Random Forest Regressor

04

Analiza techniczna: wskaźniki SMA i RSI

05

Wizualizacja wyników w aplikacji Streamlit.

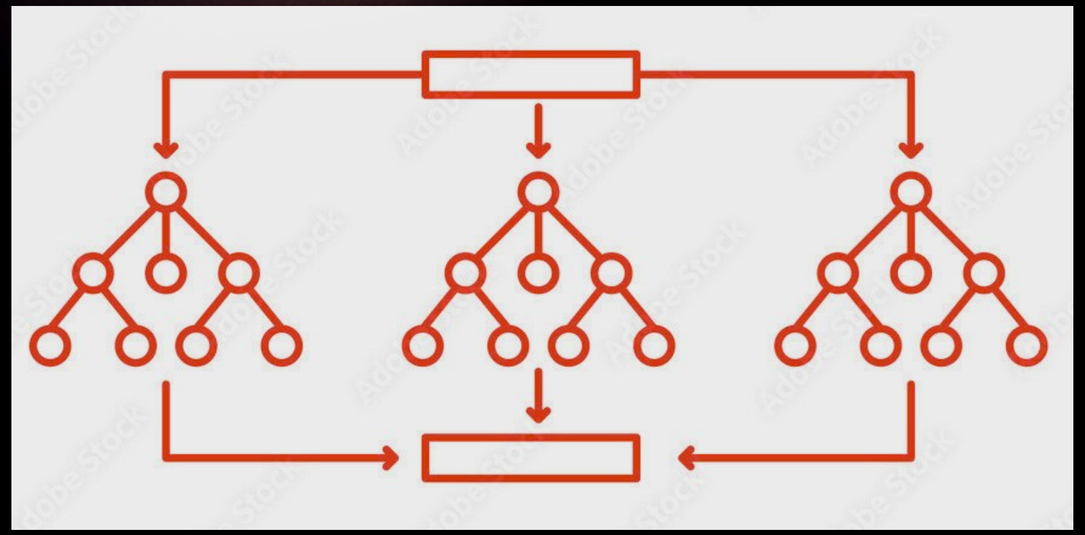
Przekształcenie danych, które występują w czasie (np. ceny akcji dzień po dniu), w taki sposób, aby model uczenia maszynowego mógł się z nich uczyć tak samo, jak z klasycznych tabel danych.

Model Random Forest

jest to model zespołowy składający się z wielu drzew decyzyjnych. Każde drzewo uczy się na losowej częci danych oraz losowym podzbiorze cech. Predykcja jest średnią wyników wszystkich drzew.

Dlaczego Random Forest?

- odporność na overfitting (przeuczenie).
- radzi sobie z nieliniowością danych giełdowych.
- pozwala ocenić ważność cech



pobieranie danych

```python
import streamlit as st
import pandas as pd
import requests
from io import StringIO

def pobierz_dane_stooq(ticker, start_date, end_date):

    url = f"https://stooq.pl/q/d/l/?s={ticker}&d1={start_date}&d2={end_date}&i=d"

    r = requests.get(url, timeout=15)

    if r.status_code != 200:
        st.error(f"HTTP {r.status_code} podczas pobierania danych")
        return None

    df = pd.read_csv(StringIO(r.text))

    return df
```

python

opóźnienia (lags)

```python
import numpy as np

def opoznienia(df, n_lags=5):

    X, y = [], []
    for i in range(n_lags, len(df)):
        X.append(df[['Open','High','Low','Close','Volume']].iloc[i-n_lags:i].values.flatten())
        y.append(df['Close'].iloc[i])

    return np.array(X), np.array(y)
```

podział danych na trening i test

```python
split = int(len(X) * 0.8)

X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

podstawowy model Random Forest

```python
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(
    n_estimators=200,       # liczba drzew
    max_depth=None,        # brak maksymalnej głębokości drzewa
    random_state=42
)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

ocena jakości modelu

```python
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

print("R2:", r2_score(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
print("MAE:", mean_absolute_error(y_test, y_pred))
```

R^2 pokazuje, jak dobrze model wyjaśnia zmienność danych – im bliżej 1, tym lepiej.

RMSE i MAE to średnie błędy predykcji – im mniejsze, tym model dokładniejszy.

strojenie modelu - GridSearchCV

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [100, 200, 400],
    "max_depth": [None, 5, 10],
    "max_features": ["sqrt", "log2"]
}

grid = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=3,
    n_jobs=-1
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_

print(grid.best_params_)
```

python

prognozowanie krokowe (iteracyjne)

```python
import numpy as np

def iterative_forecast(model, last_values, days, n_lags=5):
    current = list(last_values[-n_lags:])
    predictions = []

    for _ in range(days):
        X = np.array(current[-n_lags:]).reshape(1, -1)
        pred = model.predict(X)[0]
        predictions.append(pred)
        current.append(pred)

    return predictions
```

interpretacja sygnałów trendu

```python
# Parametry średnich kroczących i RSI
short_window = 5     # krótkoterminowa średnia SMA (np. 5 dni)
long_window  = 20    # długoterminowa średnia SMA (np. 20 dni)
rsi_period   = 14    # okno czasowe do obliczania RSI

# obliczenie średnich kroczących i RSI
df['SMA_short'] = sma(df['Close'], short_window)
df['SMA_long']  = sma(df['Close'], long_window)
df['RSI']       = rsi(df['Close'], rsi_period)

prev_short = df['SMA_short'].shift(1).iloc[i]
prev_long  = df['SMA_long'].shift(1).iloc[i]

# interpretacja trendu
if (row['SMA_short'] > row['SMA_long']) \
   and (prev_short <= prev_long) \
   and (row['RSI'] < 70):
    signal = "Trend wzrostowy"

elif (row['SMA_short'] < row['SMA_long']) \
    and (prev_short >= prev_long) \
    and (row['RSI'] > 30):
    signal = "Trend spadkowy"

else:
    signal = "Brak sygnału"
```

python

DZIĘKUJEMY ZA UWAGĘ

Maciej Otto
Laura Milczanowska