



Analiza și Prognoza Seriilor de Timp

Capitolul 8: Extensii Moderne

ARFIMA, Machine Learning, Deep Learning



Cuprins

- 1 Motivație
- 2 ARFIMA: Modele cu Memorie Lungă
- 3 Random Forest pentru Serii de Timp
- 4 LSTM: Deep Learning pentru Serii de Timp
- 5 Comparație și Selecția Modelului
- 6 Aplicații Practice

La finalul acestui capitol, veți fi capabili să:

- ① Înțelegeți conceptul de **memorie lungă** în seriile de timp
- ② Estimați și interpretați modele **ARFIMA**
- ③ Aplicați **Random Forest** pentru prognoza seriilor de timp
- ④ Construiți rețele **LSTM** pentru serii temporale
- ⑤ Comparați performanța modelelor clasice vs ML
- ⑥ Alegeți metoda potrivită în funcție de context
- ⑦ Implementați aceste metode în **Python**

Limitările Modelelor ARIMA

- Presupun **memorie scurtă**: autocorelațiile scad exponențial
- Relații liniare între variabile
- Dificultăți cu **pattern-uri complexe și neliniare**
- Necesită **staționaritate** (prin diferențiere)

Soluții Moderne

- **ARFIMA**: Captează memoria lungă (autocorelații care scad lent)
- **Random Forest**: Relații neliniare, robustețe la outlieri
- **LSTM**: Pattern-uri secvențiale complexe, dependențe pe termen lung

Când să Folosim Fiecare Metodă?

Caracteristică	ARIMA	ARFIMA	RF	LSTM
Memorie lungă	✗	✓	✓	✓
Relații neliniare	✗	✗	✓	✓
Interpretabilitate	✓	✓	~	✗
Date puține	✓	✓	✗	✗
Variabile exogene	✓	✓	✓	✓
Incertitudine	✓	✓	~	✗

Regula de Aur

Începe **simplu** (ARIMA), apoi crește complexitatea doar dacă este justificat de date și performanță.

Ce este Memoria Lungă?

Memorie Scurtă (ARMA)

- Autocorelațiile ρ_k scad **exponențial**: $|\rho_k| \leq C \cdot r^k$, $r < 1$
- Efectele șocurilor dispar **rapid**
- Sumă finită: $\sum_{k=0}^{\infty} |\rho_k| < \infty$

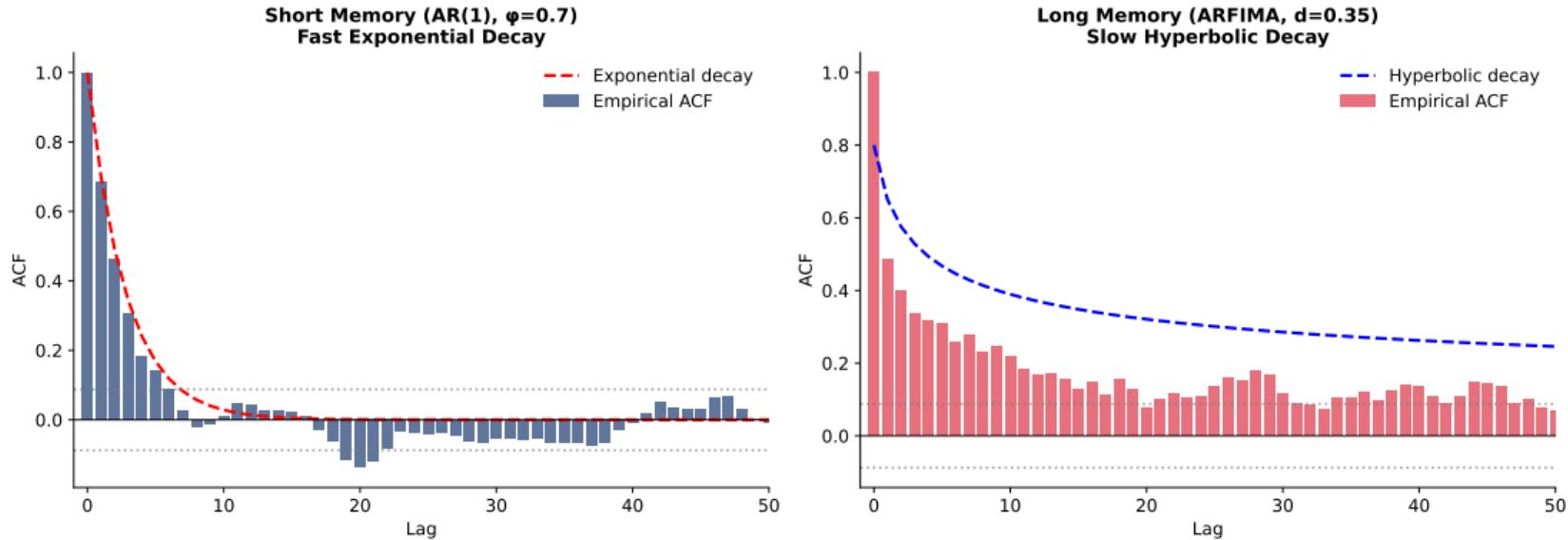
Memorie Lungă (ARFIMA)

- Autocorelațiile scad **hiperbolic**: $\rho_k \sim C \cdot k^{2d-1}$
- Efectele șocurilor persistă **mult timp**
- Sumă infinită: $\sum_{k=0}^{\infty} |\rho_k| = \infty$ (pentru $d > 0$)

Exemple cu Memorie Lungă

Volatilitatea piețelor financiare, debite râuri, trafic rețea, inflație

Comparație ACF: Memorie Scurtă vs Lungă



Stânga: AR(1) — autocorelații care scad exponențial (memorie scurtă)

Dreapta: ARFIMA cu $d = 0.35$ — autocorelații care scad hiperbolic (memorie lungă)

Modelul ARFIMA(p,d,q)

Definiție 1 (ARFIMA)

Un proces $\{Y_t\}$ urmează un model **ARFIMA(p,d,q)** dacă:

$$\phi(L)(1 - L)^d Y_t = \theta(L)\varepsilon_t$$

unde $d \in (-0.5, 0.5)$ este parametrul de **diferențiere fracționară**.

Operatorul de Diferențiere Fracționară

$$(1 - L)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-L)^k = 1 - dL - \frac{d(1-d)}{2!} L^2 - \frac{d(1-d)(2-d)}{3!} L^3 - \dots$$

- $d = 0$: ARMA standard (memorie scurtă)
- $0 < d < 0.5$: Memorie lungă, staționaritate
- $d = 0.5$: Limita staționarității
- $0.5 \leq d < 1$: Nestaționaritate, dar mean-reverting
- $d = 1$: Random walk (ARIMA standard)

Interpretarea Parametrului d

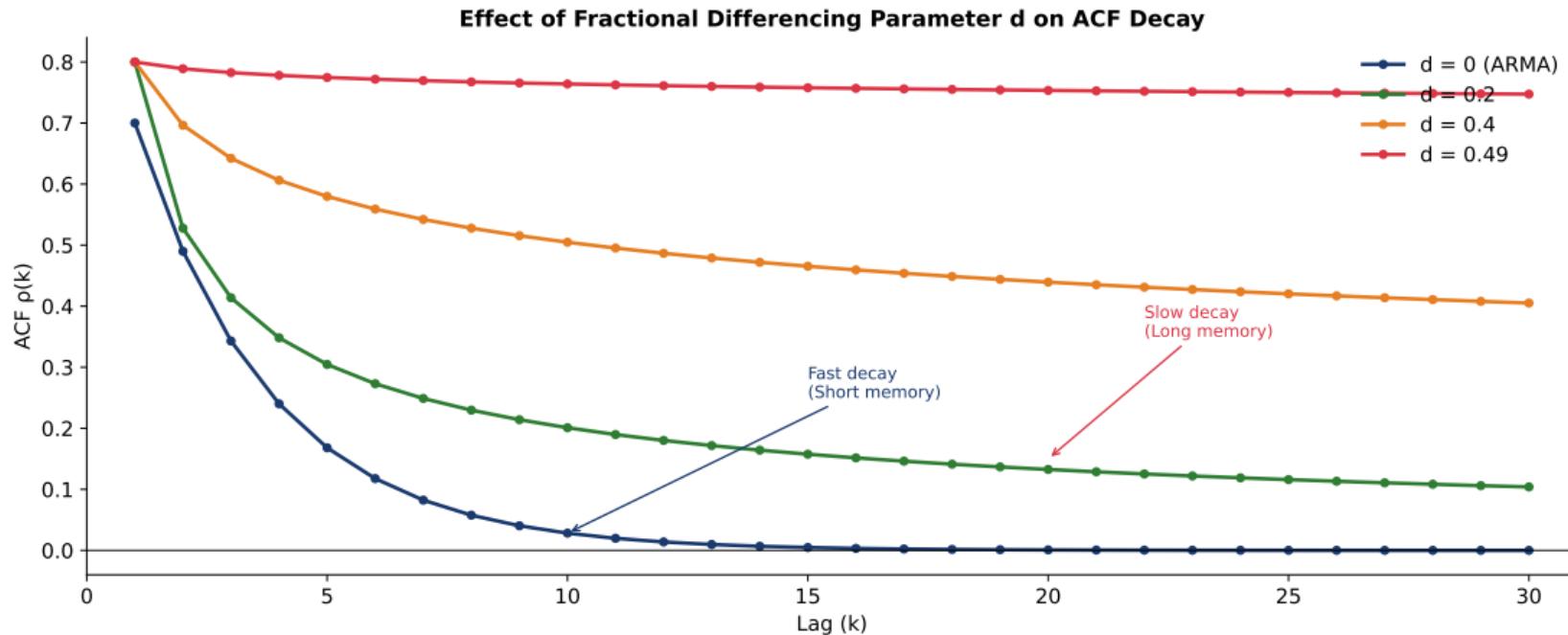
Valoare d	Comportament ACF	Interpretare
$d = 0$	Scădere exponențială	Memorie scurtă
$0 < d < 0.5$	Scădere hiperbolică	Memorie lungă, staționară
$d = 0.5$	ACF nesumabilă	La limită
$0.5 < d < 1$	Scădere foarte lentă	Memorie lungă, nestaționară
$d = 1$	ACF = 1 (constant)	Random walk

Parametrul Hurst H

Relația cu exponentul Hurst: $d = H - 0.5$

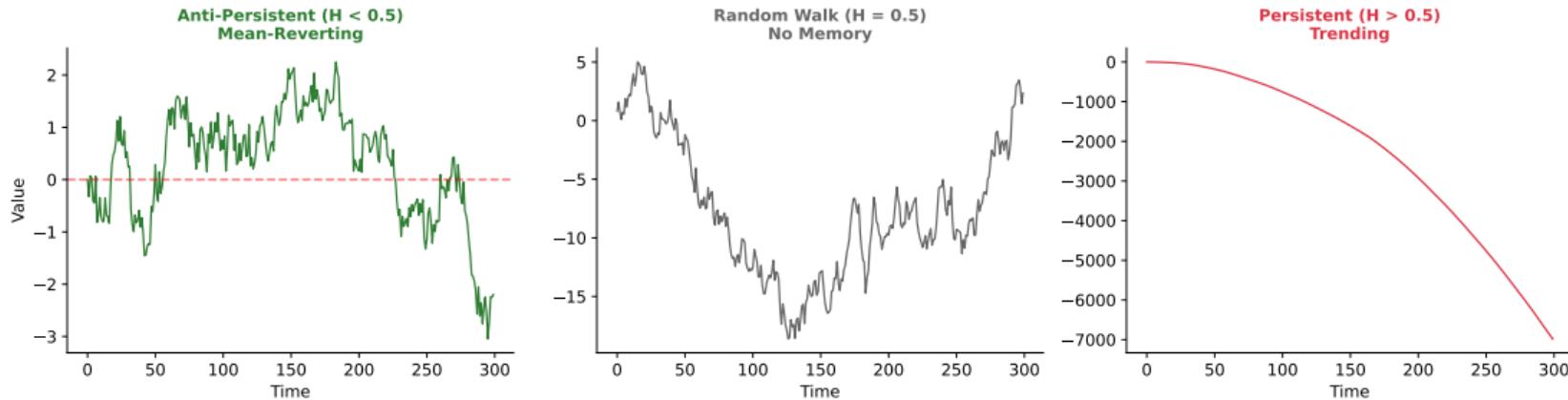
- $H = 0.5$: Mers aleator (fără memorie)
- $H > 0.5$: Persistență (trend-following)
- $H < 0.5$: Anti-persistență (mean-reverting)

Efectul Parametrului d asupra ACF



Cu cât d este mai mare, cu atât autocorelațiile scad mai lent. Pentru $d \rightarrow 0.5$, autocorelațiile rămân semnificative chiar și la lag-uri foarte mari.

Exponentul Hurst: Interpretare Vizuală



$H < 0.5$: Serie care revine frecvent la medie (mean-reverting)

$H = 0.5$: Mers aleator, imprevizibil

$H > 0.5$: Serie persistentă, tendințele continuă

Metode de Estimare

- ① **GPH (Geweke-Porter-Hudak)**: Regresie în domeniul frecvență

$$\ln I(\omega_j) = c - d \cdot \ln \left(4 \sin^2 \frac{\omega_j}{2} \right) + \varepsilon_j$$

- ② **R/S (Rescaled Range)**: Metoda lui Hurst

$$\frac{R}{S}(n) \sim c \cdot n^H$$

- ③ **MLE (Maximum Likelihood)**: Estimare completă ARFIMA

- ④ **Whittle**: Aproximare eficientă în domeniul frecvență

În Python: `arch package, statsmodels.tsa.arima.model.ARIMA cu order=(p,d,q)` unde d poate fi fracționar.

Exemplu ARFIMA în Python

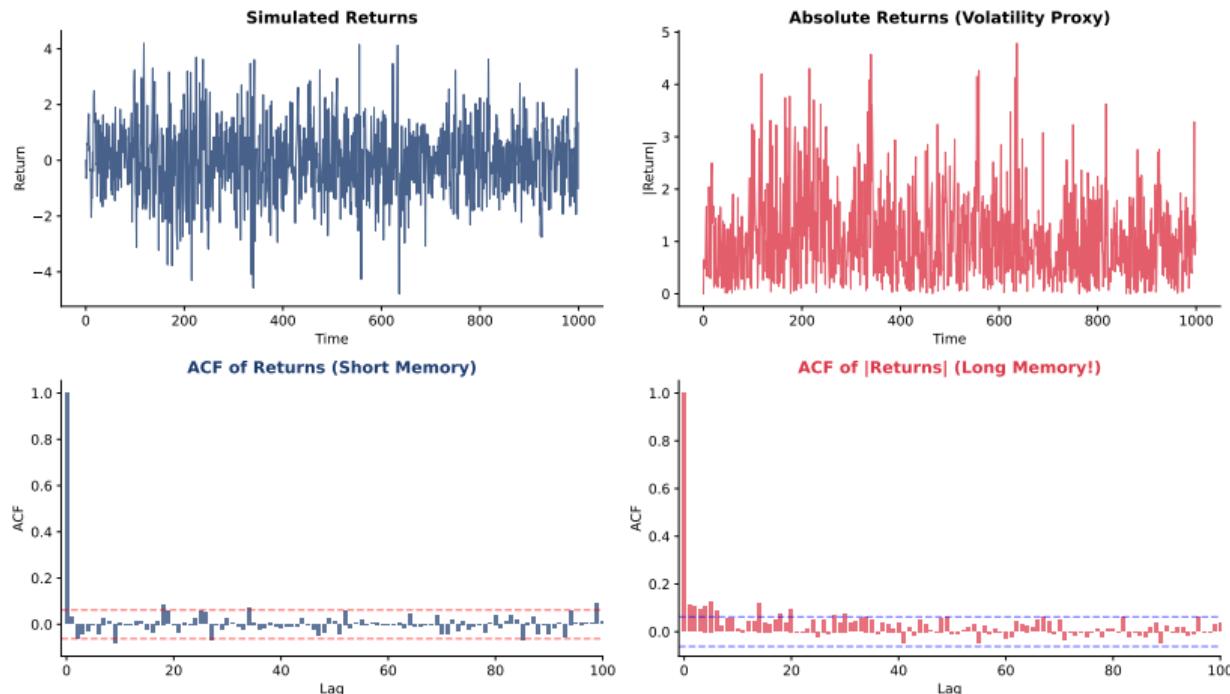
Cod Python

```
from statsmodels.tsa.arima.model import ARIMA  
model = ARIMA(y, order=(1, 0.3, 1))  
results = model.fit()
```

Notă

Estimarea ARFIMA necesită pachete specializate. În practică, se folosește adesea arch sau fracdiff în Python.

Exemplu Real: Memorie Lungă în Volatilitate



Fapt Stilizat: Randamentele financiare au memorie scurtă, dar volatilitatea ($|randamente|$) are memorie lungă! Aceasta este baza modelelor FIGARCH.

Ce este Random Forest?

- **Ansamblu** de arbori de decizie
- Fiecare arbore antrenat pe un **subset bootstrap** al datelor
- La fiecare nod, se selectează **aleator** un subset de features
- Predicția finală = **media** predicțiilor tuturor arborilor

Avantaje pentru Serii de Timp

- Captează **relații neliniare**
- **Robust** la outlieri și zgomot
- Nu necesită **staționaritate**
- Oferă **importanța features** (interpretabilitate)
- Funcționează bine cu **multe variabile**

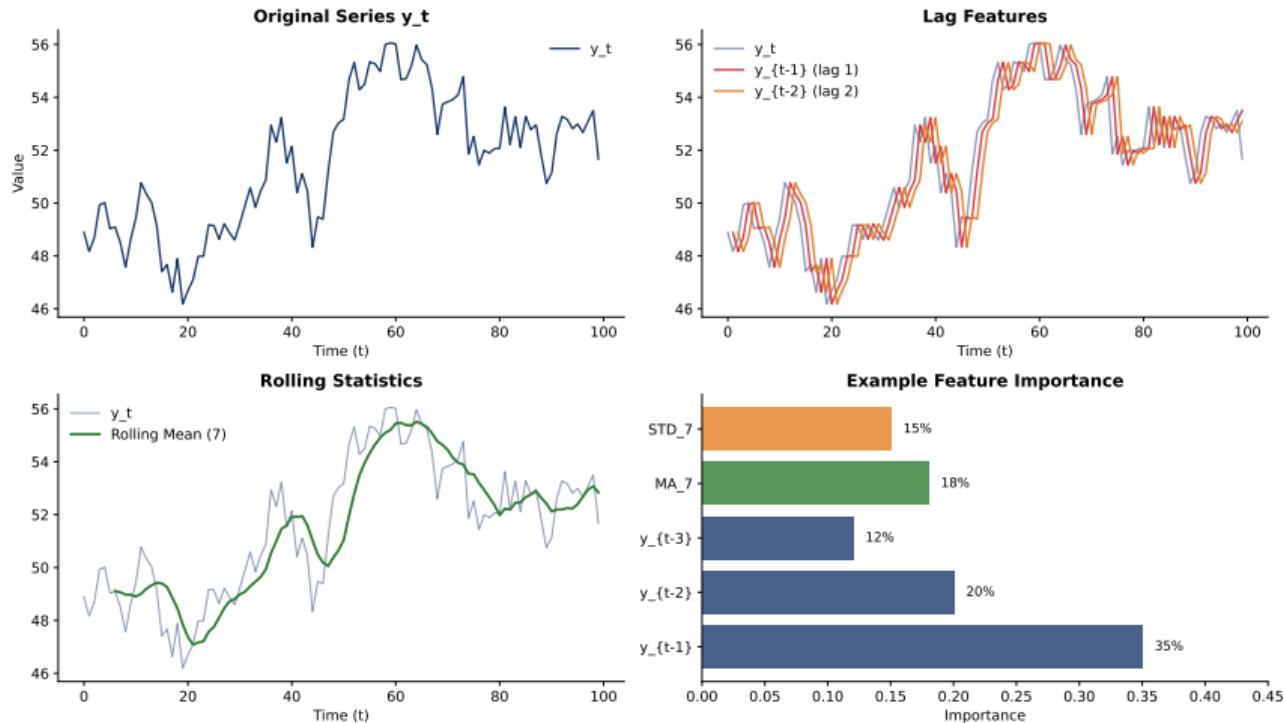
Feature Engineering pentru Serii de Timp

- ① **Lag features:** $Y_{t-1}, Y_{t-2}, \dots, Y_{t-p}$
- ② **Rolling statistics:** medie mobilă, deviație standard
- ③ **Calendar features:** ziua săptămânii, luna, sezon
- ④ **Trend features:** timp, trend pătratic
- ⑤ **Variabile exogene:** indicatori economici, evenimente

Atenție: Data Leakage!

- Nu folosi informații din viitor în features
- Train/test split: **temporal**, nu aleator!
- Rolling statistics: calculează doar pe date **anterioare**

Feature Engineering: Illustrare



Transformăm seria temporală în features: lag-uri, statistici rolling, iar modelul RF învață relațiile dintre acestea și valorile viitoare.

Random Forest: Implementare Python

Cod Python

```
from sklearn.ensemble import RandomForestRegressor  
rf = RandomForestRegressor(n_estimators=100, max_depth=10)  
rf.fit(X_train, y_train)  
predictions = rf.predict(X_test)
```

Importanța Features și Interpretare

Feature Importance

Random Forest oferă măsuri de importanță:

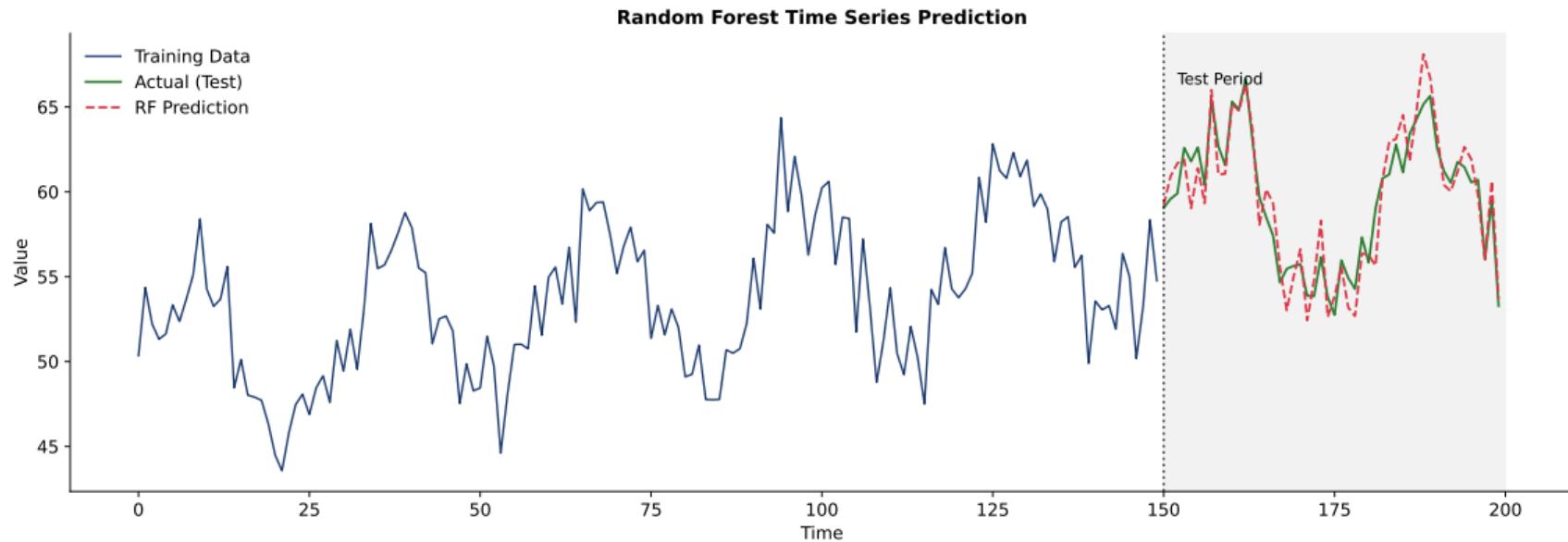
- **Mean Decrease Impurity (MDI)**: Reducerea impurității la fiecare split
- **Permutation Importance**: Cât scade performanța când feature-ul e permuatat aleator

Interpretare Tipică pentru Serii de Timp

- lag_1 foarte important \Rightarrow Autocorelare puternică
- rolling_mean important \Rightarrow Trend local contează
- month important \Rightarrow Sezonalitate prezentă

```
rf.feature_importances_ sau permutation_importance(rf, X_test, y_test)
```

Random Forest: Exemplu de Prognoză

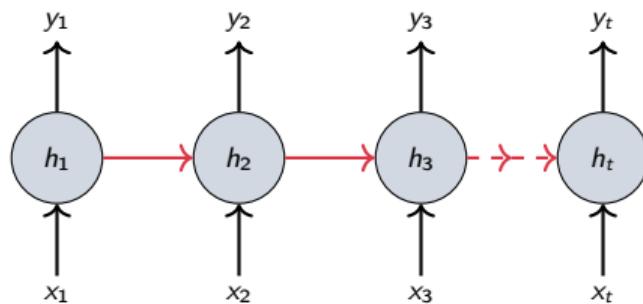


Modelul Random Forest antrenat pe date istorice (albastru) produce prognoze (roșu punctat) care urmăresc bine valorile reale din perioada de test (verde).

Rețele Neuronale Recurrente (RNN)

Ideea de Bază

- Rețele care procesează **secvențe** de date
- Au **memorie internă** (hidden state)
- Starea curentă depinde de input + starea anterioară



Problema: Vanishing Gradient

RNN simple "uită" informația din trecut îndepărtat.

Soluția LSTM

Celule speciale cu **3 porți** care controlează fluxul informației:

- **Forget Gate (f_t)**: Ce să uităm din memoria anterioară
- **Input Gate (i_t)**: Ce informație nouă să adăugăm
- **Output Gate (o_t)**: Ce să trimitem la ieșire

Ecuatiile LSTM

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{Forget})$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{Input})$$

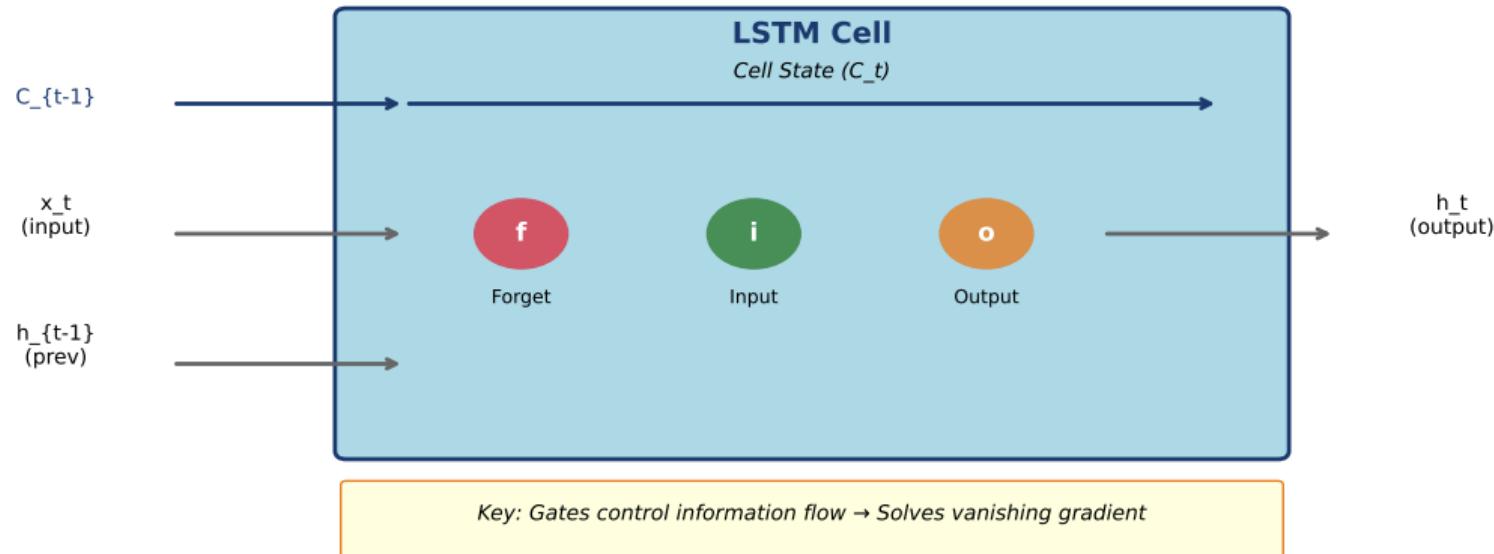
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (\text{Candidate})$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (\text{Cell state})$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{Output})$$

$$h_t = o_t \odot \tanh(C_t) \quad (\text{Hidden state})$$

Arhitectura Celulei LSTM



Porțile (forget, input, output) controlează ce informație este uitată, adăugată și transmisă. **Cell state** permite gradienților să “curgă” fără degradare.

De ce LSTM?

- Captează **dependențe pe termen lung** (spre deosebire de RNN simplu)
- Învață **pattern-uri complexe și neliniare**
- Gestionează **secvențe de lungimi variabile**
- Funcționează bine cu **date multivariate**

Dezavantaje

- Necesită **multe date** pentru antrenare
- **Computațional intensiv**
- “**Black box**” - greu de interpretat
- Sensibil la **hiperparametri**
- Poate face **overfitting** ușor

LSTM: Implementare în Python cu Keras

Cod Python

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(n, 1)),
    Dropout(0.2),
    LSTM(50),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

Pași Esențiali

- ① **Normalizare/Scalare:** MinMaxScaler sau StandardScaler
- ② **Creare secvențe:** Sliding window pentru input
- ③ **Reshape:** Format 3D (samples, timesteps, features)
- ④ **Train/Test split:** Temporal, nu aleator!

Exemplu Creare Secvențe

```
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data[i:(i + n_steps)])
    return np.array(X), np.array(y)

X, y = create_sequences(scaled_data, 10)
```

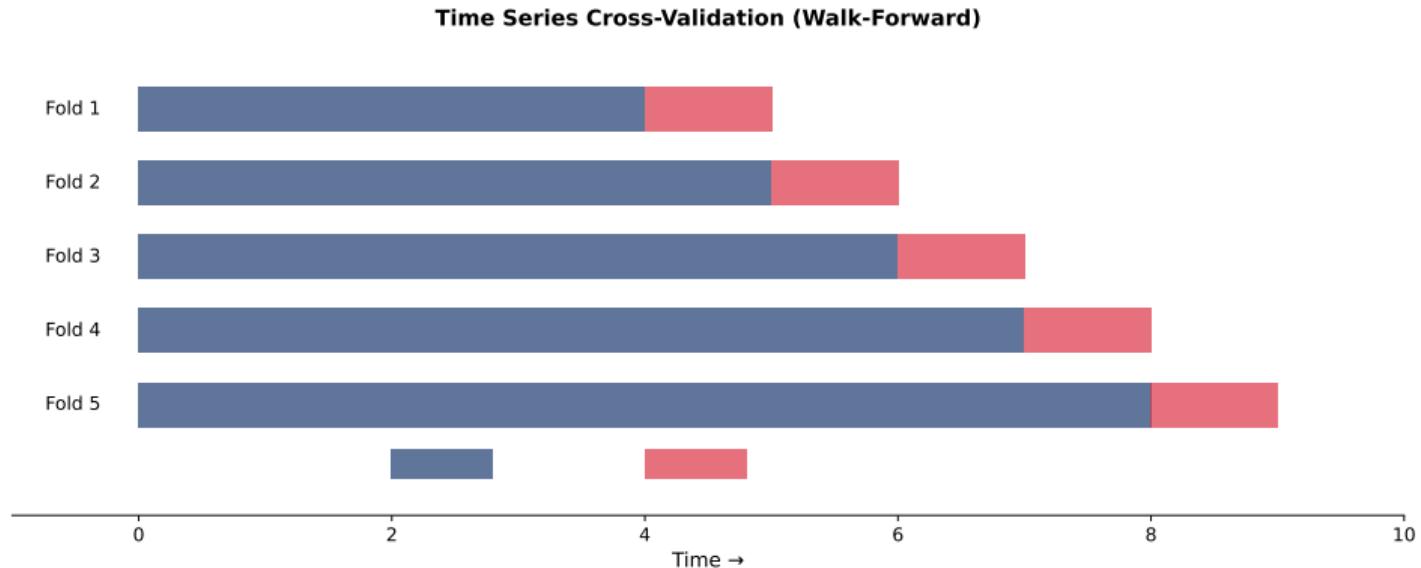
Metrici Comune

- **RMSE:** $\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$ — Eroare în unități originale
- **MAE:** $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$ — Robust la outlieri
- **MAPE:** $\frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$ — Eroare procentuală
- **MASE:** Comparat cu benchmark naiv

Validare pentru Serii de Timp

- **Nu** folosiți cross-validation standard!
- Folosiți **Time Series Cross-Validation** (walk-forward)
- Sau **train/validation/test** split temporal

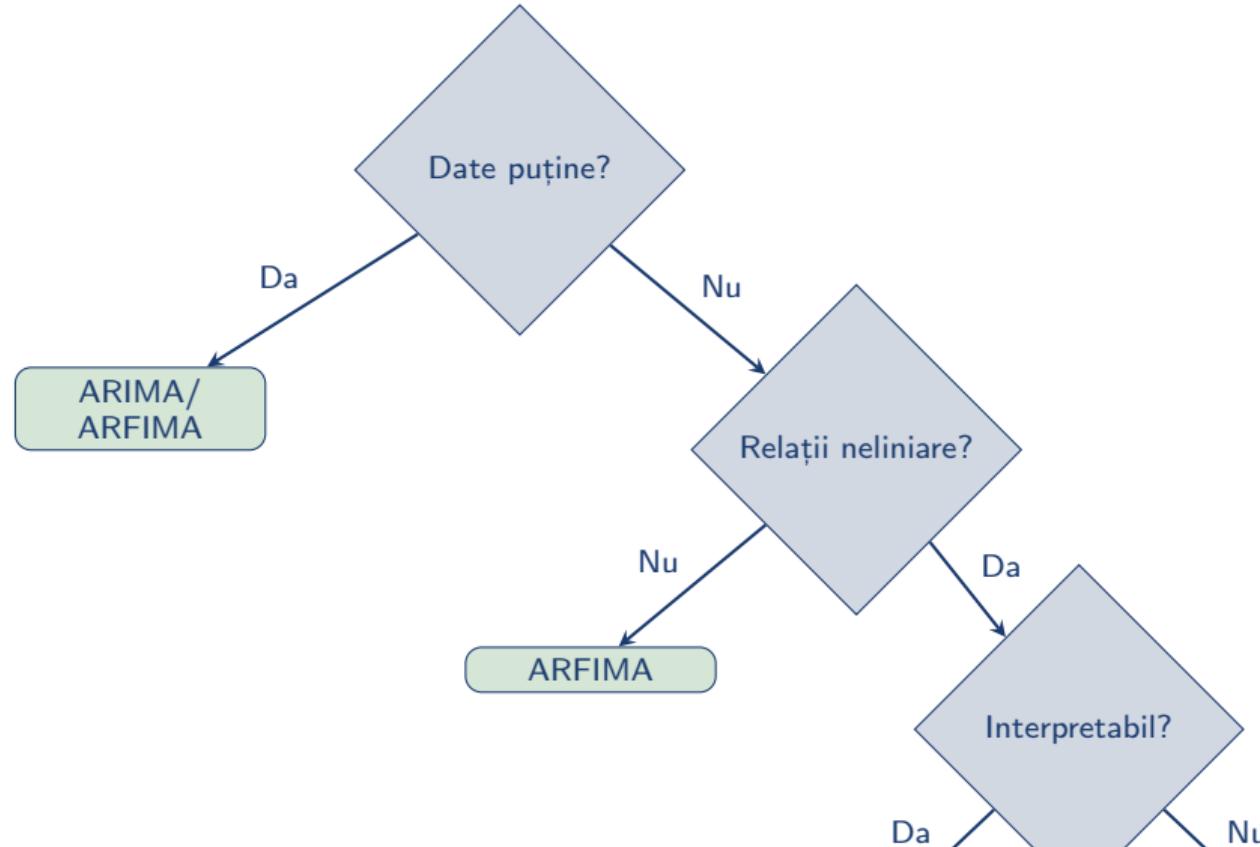
Time Series Cross-Validation



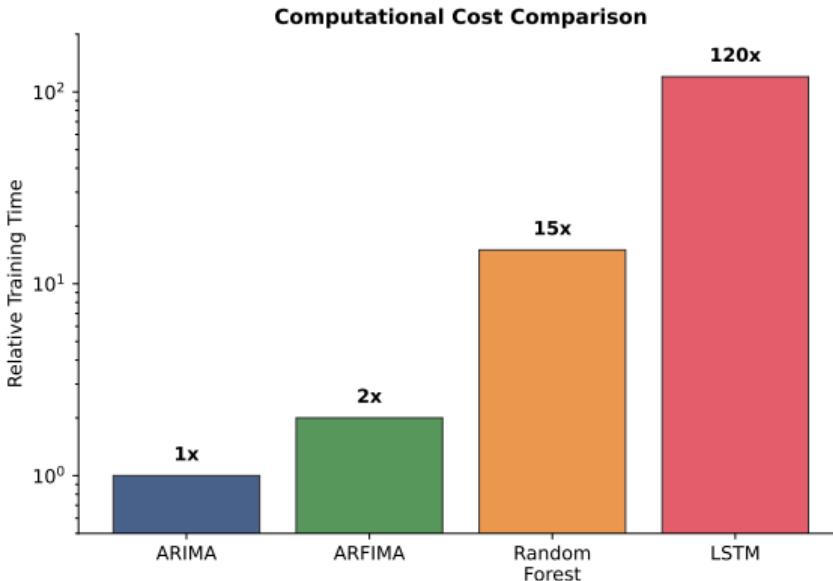
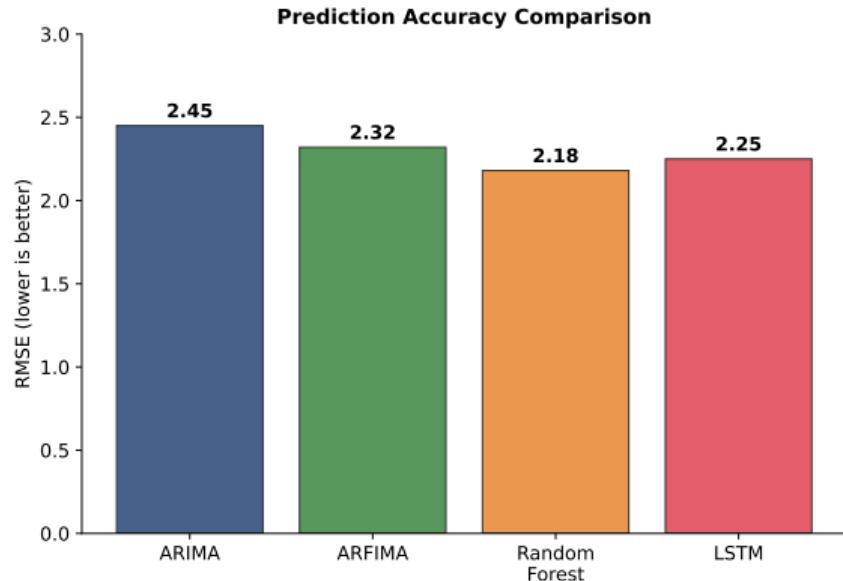
Implementare Python

```
from sklearn.model_selection import TimeSeriesSplit  
tscv = TimeSeriesSplit(n_splits=5)
```

Ghid de Selectie a Modelului



Comparație Modele: Acuratețe vs Cost Computațional



Trade-off: Modelele ML pot avea acuratețe ușor mai bună, dar costul computațional crește semnificativ. Pentru date puține sau interpretabilitate, ARIMA/ARFIMA rămân alegeri excelente.

De ce Bitcoin?

- Volatilitate **extremă** și pattern-uri complexe
- Potențială **memorie lungă** în volatilitate
- Relații **neliniare** cu variabile exogene
- Date disponibile la **frecvență înaltă**

Abordare Comparativă

- ❶ ARIMA pe randamente
- ❷ ARFIMA pentru memorie lungă
- ❸ Random Forest cu features tehnice
- ❹ LSTM pe secvențe de prețuri

Caracteristici

- **Sezonalitate multiplă:** zilnică, săptămânală, anuală
- **Tendință de creștere** pe termen lung
- **Variabile exogene:** temperatură, zi liberă, preț
- **Anomalii:** evenimente speciale, defecțiuni

Provocări

- Pattern-uri la scale temporale diferite
- Interacțiuni complexe între variabile
- Necesitatea prognozelor pe orizonturi diferite

Formule Cheie – Rezumat

ARFIMA(p,d,q)

$$\phi(L)(1 - L)^d Y_t = \theta(L)\varepsilon_t$$

$d \in (-0.5, 0.5)$: memorie lungă

LSTM Cell

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Forget, Input, Output gates

Memorie Lungă

ACF: $\rho_k \sim C \cdot k^{2d-1}$

Hurst: $d = H - 0.5$

$H > 0.5$: persistență

Metrici Evaluare

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum (y_i - \hat{y}_i)^2}$$

$$\text{MAPE} = \frac{100}{n} \sum \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Random Forest

$$\hat{y} = \frac{1}{B} \sum_{b=1}^B T_b(x)$$

B arbori, features aleatorii

Time Series CV

Walk-forward validation

Train → Test (temporal split)

Ce am învățat

- **ARFIMA:** Extinde ARIMA pentru memorie lungă (d fracționar)
- **Random Forest:** Ansamblu de arbori, relații neliniare, interpretabil
- **LSTM:** Deep learning pentru secvențe, dependențe complexe
- **Trade-offs:** Complexitate vs interpretabilitate vs date necesare

Recomandări Practice

- Începe cu modele **simple** (ARIMA) ca baseline
- Folosește **Time Series CV** pentru evaluare corectă
- ML necesita **feature engineering** atent
- LSTM: doar cu **multe date și resurse computaționale**

Quiz Rapid

- ① Ce semnifică $d = 0.3$ într-un model ARFIMA?
- ② De ce folosim Time Series CV în loc de k-fold standard?
- ③ Care este avantajul principal al LSTM față de RNN simplu?
- ④ Ce tip de model ai alege pentru date puține și relații liniare?
- ⑤ Ce înseamnă "data leakage" în contextul ML pentru serii de timp?

Răspunsuri Quiz

- ❶ $d = 0.3$: Memorie lungă, seria este staționară dar autocorelațiile scad lent (hiperbolic). Persistență moderată.
- ❷ **Time Series CV**: Pentru a respecta ordinea temporală. K-fold standard ar folosi date viitoare pentru a prezice trecutul (data leakage).
- ❸ **LSTM vs RNN**: LSTM rezolvă problema “vanishing gradient” prin mecanismul de porți, permitând învățarea dependențelor pe termen lung.
- ❹ **Date puține, relații liniare**: ARIMA sau ARFIMA. ML necesită multe date pentru a generaliza bine.
- ❺ **Data leakage**: Folosirea informației din viitor în features sau în antrenare. Ex: calcularea mediei mobile folosind și date viitoare, sau k-fold standard care amestecă ordinea temporală.

Extensii și Subiecte Avansate

- **Transformer** pentru serii de timp (Temporal Fusion Transformer)
- **Prophet** (Facebook/Meta) pentru sezonalitate
- **Neural Prophet**: Prophet + rețele neuronale
- **Ensemble methods**: Combinarea mai multor modele
- **Anomaly detection** cu ML

Întrebări?