Minor in AI

Sequence Modelling

Auto-Regressive Models & Text Processing

March 04, 2025

1 Introduction

Time series forecasting is an essential area of statistical modeling. Auto-Regressive (AR) models are a fundamental class of models for predicting future values based on past observations. In this document, we discuss AR(p) models, explore AR(1) in detail, and link them to Markov Chains. We will also compare sequences that fit into these models and analyze how they differ.

2 Auto-Regressive (AR) Models

An **Auto-Regressive (AR)** model of order p, denoted as $AR(p)$, is a time series model where the current value depends linearly on its past p values:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \varepsilon_t$$

where ϕ_i are the model parameters and ε_t is white noise.

2.1 The AR(1) Model

A special case is $AR(1)$:

$$X_t = \phi X_{t-1} + \varepsilon_t$$

which implies that the current value only depends on the immediate past value.

2.2 Connection to Markov Chains

An $AR(1)$ model can be linked to a **first-order Markov process**, where the next state only depends on the current state.

A Markov Chain is defined by transition probabilities:

$$P(X_{t+1}|X_t)$$

and can be represented using a **transition matrix**.

3 Sequence Data

Sequence data refers to ordered data points collected over time or space, where the order of occurrence is crucial for understanding patterns, dependencies, and predictions. It is widely used in various domains, such as time-series analysis, natural language processing (NLP), and bioinformatics.

3.1 Characteristics of Sequence Data

- **Ordered Structure** – Each element in the sequence follows a specific order, making the position of data points essential.
- **Temporal or Spatial Dependency** – The value of a data point is often dependent on previous values.
- **Variable Length** – Some sequences have fixed lengths (e.g., DNA sequences), while others can be dynamic (e.g., financial stock prices).

3.2 Types of Sequence Data

1. **Time-Series Data**
   A series of data points indexed in time order.
   Example: Daily stock prices, weather temperature recordings.

2. **Text Sequences**
   A sequence of words, sentences, or characters used in NLP.
   Example: Sentences in a paragraph.

3. **Genomic Sequences**
   A sequence of nucleotides (A, T, G, C) representing DNA.

4. **Sequential Clickstream Data**
   User interactions on a website over time.

5. **Event-based Sequences**
   Logs of user activities or system operations in chronological order.

Example:

A temperature recording over 10 days:

$$T = 30.5, 31.2, 29.8, 30.1, 30.7, 31.0, 29.9, 30.3, 30.8, 31.1$$

Time series can be categorized into:

- **Autoregressive Sequences:** Depend on past values (e.g., AR models).
- **Markovian Sequences:** Depend only on the current state.

Example: Stock prices follow an AR process, whereas a random walk is a Markov process.

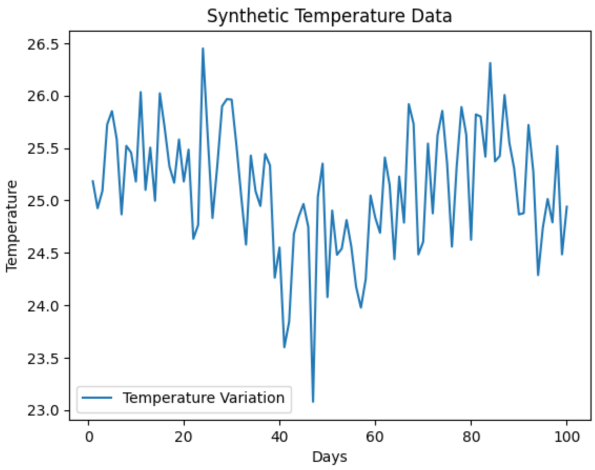
4 Code Implementation

4.1 Generating a Synthetic Time Series

We first generate a synthetic temperature dataset that exhibits seasonal patterns with random noise.

```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(182)

# Generate days and temperature data with seasonal variation and noise
days = np.arange(1,101)
temperature = 25 + 0.5 * np.sin(0.1*days) + np.random.normal(0,0.5,size
    =100)

# Plot the temperature data
plt.plot(days, temperature, label='Temperature Variation')
plt.xlabel('Days')
```

```
plt.ylabel('Temperature')
plt.title('Synthetic Temperature Data')
plt.legend()
plt.show()
```



4.2 Creating AR(p) Model Inputs

To build an AR(p) model, we construct feature vectors from past observations.

```
p = 3  # Number of past observations to use

# Prepare feature matrix X and target vector y
X, y = [], []
for i in range(len(temperature) - p):
    X.append(temperature[i:i+p])  # Take past 'p' values
    y.append(temperature[i+p])  # Target value is the next observation

X, y = np.array(X), np.array(y)

print("Feature matrix shape:", X.shape)
print("Target vector shape:", y.shape)
```

```
Feature matrix shape: (97, 3)
Target vector shape: (97,)
```

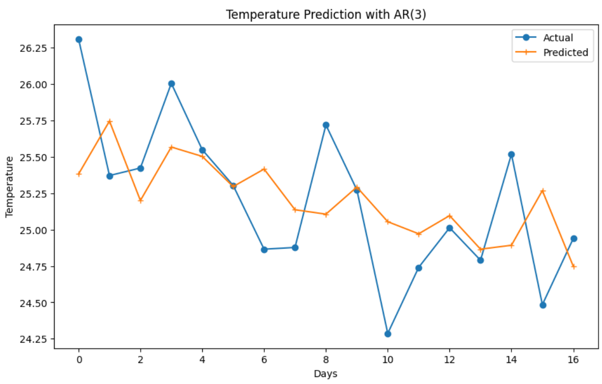
4.3 Training an AR(3) Model

We use the least squares method to estimate the coefficients of an AR(3) model.

```
from sklearn.metrics import mean_squared_error

# Splitting data into training and testing sets
X_train, X_test = X[:80], X[80:]
y_train, y_test = y[:80], y[80:]

# Add bias term for intercept
X_train_with_bias = np.c_[np.ones(X_train.shape[0]), X_train]
X_test_with_bias = np.c_[np.ones(X_test.shape[0]), X_test]

```

```python
# Compute least squares solution
coeff = np.linalg.lstsq(X_train_with_bias, y_train, rcond=None)[0]
print("Estimated coefficients:", coeff)

# Predict on test data
y_pred = X_test_with_bias @ coeff
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Plot actual vs. predicted values
plt.figure(figsize=(10,6))
plt.plot(range(len(y_test)), y_test, label='Actual', marker='o')
plt.plot(range(len(y_pred)), y_pred, label='Predicted', marker='+')
plt.xlabel("Days")
plt.ylabel("Temperature")
plt.title("Temperature Prediction with AR(3)")
plt.legend()
plt.show()
```

```
Estimated coefficients: [10.36153868  0.29757698 -0.08728726  0.37726841]
Mean Squared Error: 0.21713134375598753
```



4.4 Linking AR(1) to Markov Chains

An AR(1) model can be rewritten as:

$$X_t = \phi X_{t-1} + \epsilon_t \tag{1}$$

This form shows that the next state depends only on the current state, similar to a first-order Markov Chain.

```python
from sklearn.linear_model import LinearRegression

X_ar1 = temperature[:-1].reshape(-1,1)  # AR(1) uses only previous value
y_ar1 = temperature[1:]

# Split data into training and testing
train_X_ar1, test_X_ar1 = X_ar1[:80], X_ar1[80:]
train_y_ar1, test_y_ar1 = y_ar1[:80], y_ar1[80:]

# Fit linear regression model for AR(1)
```

```python
model = LinearRegression()
model.fit(train_X_ar1, train_y_ar1)

y_pred_ar1 = model.predict(test_X_ar1)
mse_ar1 = mean_squared_error(test_y_ar1, y_pred_ar1)
print("Mean Squared Error for AR(1):", mse_ar1)
```

```
Mean Squared Error for AR(1): 0.2343436817353426
```

4.5 Markov Chain Transition Probabilities

We now convert the temperature values into discrete states and compute the transition probabilities.

```python
from collections import defaultdict

discrete_temp = np.round(temperature).astype(int)
transition_counts = defaultdict(lambda: defaultdict(int))

# Count transitions from one state to another
for i in range(len(discrete_temp) - 1):
    transition_counts[discrete_temp[i]][discrete_temp[i+1]] += 1

# Normalize to get probabilities
transition_matrix = {}
for state, transitions in transition_counts.items():
    total = sum(transitions.values())
    for next_state, count in transitions.items():
        transition_matrix[(state, next_state)] = count / total

print("Transition Probabilities:", transition_matrix)
```

```
Transition Probabilities:
{(25, 25): 0.5614035087719298,
(25, 26): 0.2982456140350877,
(25, 24): 0.12280701754385964,
(25, 23): 0.017543859649122806,
(26, 26): 0.41379310344827586,
(26, 25): 0.5172413793103449,
(26, 24): 0.06896551724137931,
(24, 25): 0.75, (24, 24): 0.25, (23, 25): 1.0}
```

4.6 Autoregressive Model: Influence of Parameter

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# generate synthetic temperature data
days = np.arange(1, 366)
temperature = 25 + 0.5 * np.sin(0.1 * days) + np.random.normal(0, 0.5,
    size=365)

```

```python
# We create a function to generate input-output pairs for
# training the AR model. The function takes a time series
# and extracts sequences of length \textit{p} as inputs,
# with the next value as the output.

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

We train AR models with different values of p and compute the mean squared error for each.

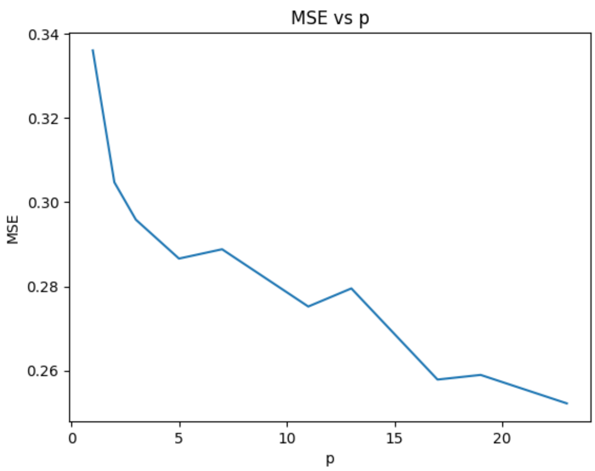
```python
# define training size
training_size = 280

# different values of p to evaluate
p_values = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]

ar_models = {}
mse_scores = {}

for p in p_values:
    X, y = prepare_data(temperature, p)

    # Split data into training and testing sets
    X_train, X_test = X[:training_size], X[training_size:]
    y_train, y_test = y[:training_size], y[training_size:]

    # Train linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Compute MSE
    mse = mean_squared_error(y_test, y_pred)

    # Store model and error
    ar_models[p] = model
    mse_scores[p] = mse

# Print MSE scores
print(mse_scores)
```

```
{1: 0.33608976834741705, 2: 0.30475085893549164, 3: 0.2958559235325939,
5: 0.28662182132656844, 7: 0.28883656538341396, 11: 0.27522790581420287,
13: 0.2795310851246623, 17: 0.25787422226688395, 19: 0.2589797781277694,
23: 0.2522163052171486}
```

Finally, we plot the MSE values against different p values to observe how increasing the number of past values affects prediction accuracy.

```python
# Plot MSE vs p
plt.plot(list(mse_scores.keys()), list(mse_scores.values()), marker='o')
```

```
plt.xlabel("p")
plt.ylabel("MSE")
plt.title("MSE vs p")
plt.show()
```



5 Text Processing

Text processing is an essential step in Natural Language Processing (NLP) and data cleaning. Raw text data often contains inconsistencies such as punctuation, whitespace, and case variations, which need to be addressed before analysis. One common preprocessing technique is tokenization, which involves splitting text into meaningful units called tokens.

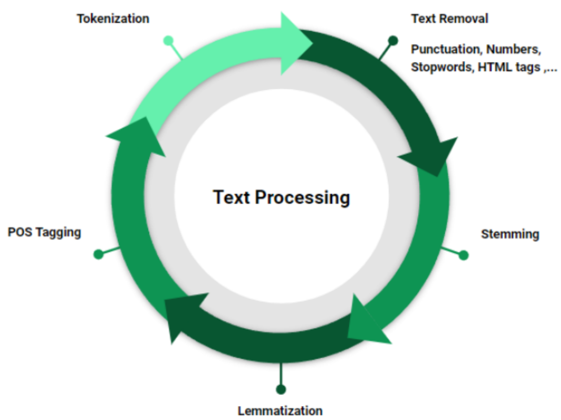


Figure 1: Steps of Text Processing

In this example, we preprocess the text from the novel `The_Time_Machine` by H.G. Wells. The following steps are involved:

- Convert text to lowercase to maintain consistency.
- Remove punctuation and whitespace to standardize the content.
- Split the text into individual words (tokens) for further processing.

The following Python code reads the text file, preprocesses the text, and performs tokenization.

5.1 Reading the Text File

The first step is to read the file and store its content as a string.

```
# Read the file
txt_filename = "The_Time_Machine.txt"

with open(txt_filename, "r", encoding="utf-8") as f:
    lines = f.read()
```

5.2 Converting to Lowercase

Converting text to lowercase ensures uniformity in token processing.

```
import re

# Convert text to lowercase
lines = lines.lower()
```

5.3 Removing Punctuation and Whitespace

Special characters and extra whitespace are removed using regular expressions.

```
# Remove punctuation and whitespace
lines = re.sub(r"[\s\n]", "", lines)
```

5.4 Analyzing the Processed Text

After preprocessing, we can examine the length of the processed text and extract a sample.

```
# Get the length of the processed text
len(lines)
```

```
0
```

```
# Display the last 10 characters
lines[-10:]
```

''

5.5 Tokenization

Finally, we split the text into individual tokens using whitespace as a delimiter.

```
# Tokenize the text
tokens = lines.split()
```

6 Key Takeaways

1. **Auto-Regressive (AR) Models**: AR models predict future values based on past observations. An AR(p) model uses p past values for forecasting.

2. **AR(1) and Markov Chains**: The AR(1) model exhibits a first-order dependence similar to Markov Chains, where the next state depends only on the current state.

3. **Sequence Data Characteristics**: Time series and sequential data exhibit ordered structure, temporal dependencies, and varying lengths, making them crucial in forecasting and pattern recognition.

4. **Synthetic Data Generation**: We created a synthetic temperature dataset to illustrate AR model applications in real-world time series analysis.

5. **Feature Engineering for AR Models**: Constructing input matrices with past observations enables model training for time series forecasting.

6. **Least Squares Estimation for AR Models**: Coefficients of AR models can be estimated using least squares, enabling effective forecasting of future values.

7. **Model Performance**: Evaluating predictions using metrics like Mean Squared Error (MSE) helps assess model accuracy in time series forecasting.

8. **Text Processing Fundamentals**: Text data needs preprocessing steps such as tokenization, stemming, lemmatization, and stopword removal to improve analysis and modeling.

Google Collab Link : Code Here!!!