Time Series: 1st lesson – Linear Regression With Time Series

Forecasting is perhaps the most common application of machine learning in the real world. Businesses forecast product demand, governments forecast economic and population growth, meteorologists forecast the weather. The understanding of things to come is a pressing need across science, government, and industry (not to mention our personal lives!), and practitioners in these fields are increasingly applying machine learning to address this need.

Time series forecasting is a broad field with a long history. This course focuses on the application of modern machine learning methods to time series data with the goal of producing the most accurate predictions. The lessons in this course were inspired by winning solutions from past Kaggle forecasting competitions but will be applicable whenever accurate forecasts are a priority.

Time series:

A set of observations recorded over time. In forecasting applications, the observations are typically recorded with a regular frequency, like daily or monthly.

import pandas as pd

df = pd.read\_csv(

"../input/ts-course-data/book\_sales.csv",

index\_col='Date',

parse\_dates=['Date'],

).drop('Paperback', axis=1)

df.head()

Hardcover

Date

2000-04-01 139

2000-04-02 128

2000-04-03 172

2000-04-04 139

2000-04-05 191

This series records the number of hardcover book sales at a retail store over 30 days. Notice that we have a single column of observations Hardcover with a time index Date.

Linear regression with time series:

Linear regression is widely used in practice and adapts naturally to even complex forecasting tasks, which designed for constructing forecasting models. The linear regression algorithm learns how to make a weighted sum from its input features. For two features include:

target = weight\_1 \* feature\_1 + weight\_2 \* feature\_2 + bias

During training, the regression algorithm learns values for the parameters weight\_1, weight\_2, and bias that best fit the target. (This algorithm is often called *ordinary least* squares since it chooses values that minimize the squared error between the target and the predictions.) The weights are also called *regression coefficients* and the bias is also called the intercept because it indicates where the graph of this function crosses the y-axis.

Time-step features:

Time-step features are features that can derive directly from the time index. The most basic time-step feature is the time dummy, which counts off time steps in the series from beginning to end.

import numpy as np

df['Time'] = np.arange(len(df.index))

df.head()

Hardcover Time

Date

2000-04-01 139 0

2000-04-02 128 1

2000-04-03 172 2

2000-04-04 139 3

2000-04-05 191 4

Linear regression with the time dummy produces the model:

target = weight \* time + bias

The time dummy then lets us fit curves to time series in a time plot, where Time forms the x-axis.

import matplotlib.pyplot as plt

import seaborn as sns

plt.style.use("seaborn-whitegrid")

plt.rc(

"figure",

autolayout=True,

figsize=(11, 4),

titlesize=18,

titleweight='bold',

)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=16,

titlepad=10,

)

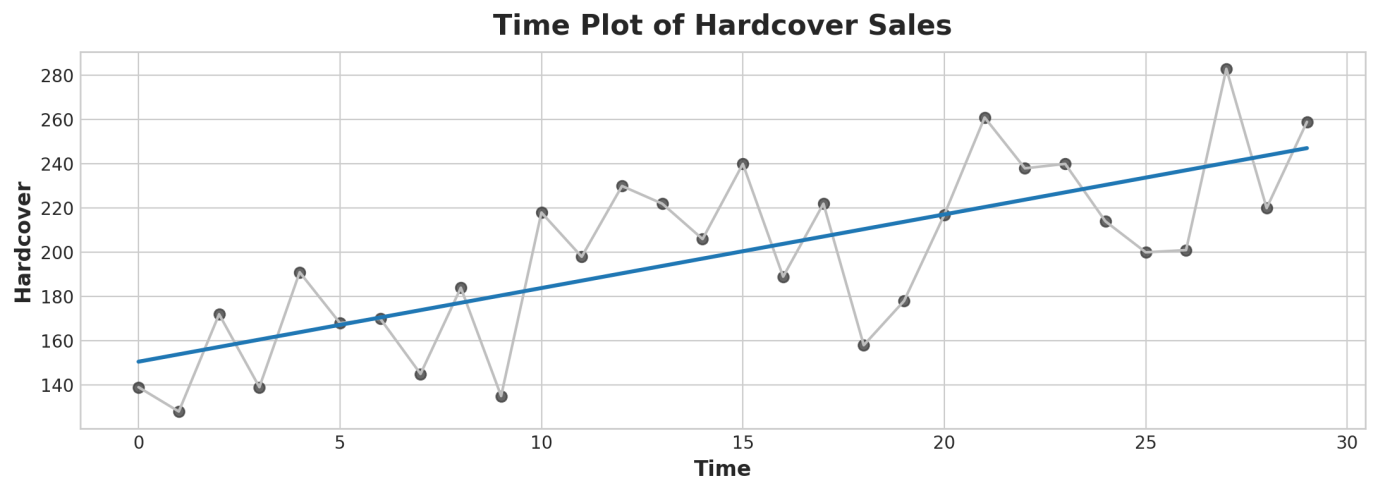
%config InlineBackend.figure\_format = 'retina'

fig, ax = plt.subplots()

ax.plot('Time', 'Hardcover', data=df, color='0.75')

ax = sns.regplot(x='Time', y='Hardcover', data=df, ci=None, scatter\_kws=dict(color='0.25'))

ax.set\_title('Time Plot of Hardcover Sales');



Time-step features let you model time dependence. A series is time dependent if its values can be predicted from the time they occured. In the *Hardcover Sales* series, we can predict that sales later in the month are generally higher than sales earlier in the month.

Lag features:

To make a lag feature, shift the observations of the target series so that they appear to have occured later in time. Here we've created a 1-step lag feature, though shifting by multiple steps is possible too.

df['Lag\_1'] = df['Hardcover'].shift(1)

df = df.reindex(columns=['Hardcover', 'Lag\_1'])

df.head()

Hardcover Lag\_1

Date

2000-04-01 139 NaN

2000-04-02 128 139.0

2000-04-03 172 128.0

2000-04-04 139 172.0

2000-04-05 191 139.0

Linear regression with a lag feature produces the model:

target = weight \* lag + bias

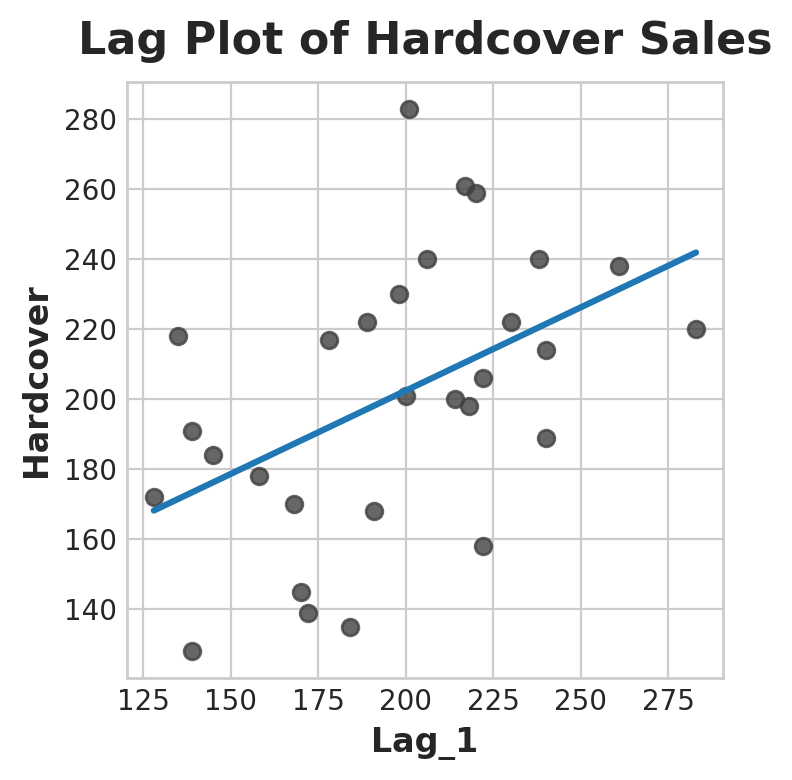
So lag features let us fit curves to lag plots where each observation in a series is plotted against the previous observation.

fig, ax = plt.subplots()

ax = sns.regplot(x='Lag\_1', y='Hardcover', data=df, ci=None, scatter\_kws=dict(color='0.25'))

ax.set\_aspect('equal')

ax.set\_title('Lag Plot of Hardcover Sales');



You can see from the lag plot that sales on one day (Hardcover) are correlated with sales from the previous day (Lag\_1). When you see a relationship like this, you know a lag feature will be useful.

More generally, lag features let you model serial dependence. A time series has serial dependence when an observation can be predicted from previous observations. In Hardcover Sales, we can predict that high sales on one day usually mean high sales the next day.

Adapting machine learning algorithms to time series problems is largely about feature engineering with the time index and lags. For most of the course, we use linear regression for its simplicity, but these features will be useful whichever algorithm you choose for your forecasting task.