Everything

Logan Bontrager

December 12, 2020

Introduction

The intent of the project was to explore bayesian time series analysis through autoregressive models. Autoregressive models or AR were introduced in the 1980's to capture casual relationships between macroeconomic variables and have since progressed to be an industry benchmark for time series analysis. The following explores autoregressive models and their bayesian counterparts. Note that for this project, all time series model are self-implemented while all statistical tests are sourced from the statsmodel python package.

Autoregression

The autoregressive model assumes that there is a relationship between a vector of observations and a given number of lagged data points. Thus, the model formulates the current times series values as a linear combination of p past observations. This is given by

$$y_t = \beta_0 + \beta_1 y_{t-1} + ... + \beta_{t-p} y_{t-p} + u_t$$

where β is the weight vector, $Y = (y_t, y_{t-1}, ..., y_1)$ represents the vector of observations, and U is the error vector.

Bayesian Autoregression

For the Bayesian approach, I augmented the model using a simple prior, $\beta \sim N(0, \frac{1}{\alpha} I)$, and learned the parameters for the posterior distribution using the evidence approximation and the iterative algorithm from the book (eq. 3.91, 3.92, and 3.95). Then, we can use these values to find predictions for unobserved values with the maximum a posterior and compute an error value.

Implementation

For this project, I will be modeling from the original FANG stock group, Facebook, Amazon, Netflix, and Google. I sourced my data from the Alpha Vantage

API. This report will only cover the Amazon ticker in sake of conciseness. The series themselves consist of daily closing price for each respective stock from the last 8 years.

An important concept in time series analysis with the VAR model is stationarity. Stationarity in academic litature is divided into weak and strict stationarity. For our context, we will be using weak stationarity, or the case where a time series's mean, variance, and correlation doesn't vary with time. Strict stationarity refers to when the joint distribution of observations is consistent over time.

A common test used to check for time series stationarity is the Augmented Dickey Fuller test, ADF. The test assesses the pressence of a unit root in the data or a stochastic trend that could lead us to believe that the underlying distribution is not consistent over time. ADF is given by,

$$y_t = c + \beta t + \alpha y_{t-1} + \phi_1 \Delta Y_{t-2} + ... + \phi_p \Delta Y_{t-p} + u_t$$

In addition, we also need to do exploratory analysis on the series autocorrelation or correlation of the series with itself. This will allow us to determine if the data has seasonality and give us insight into how to properly transform it for the model. The autocorrelation and partial aurocorrelation function are described below for times s and t.

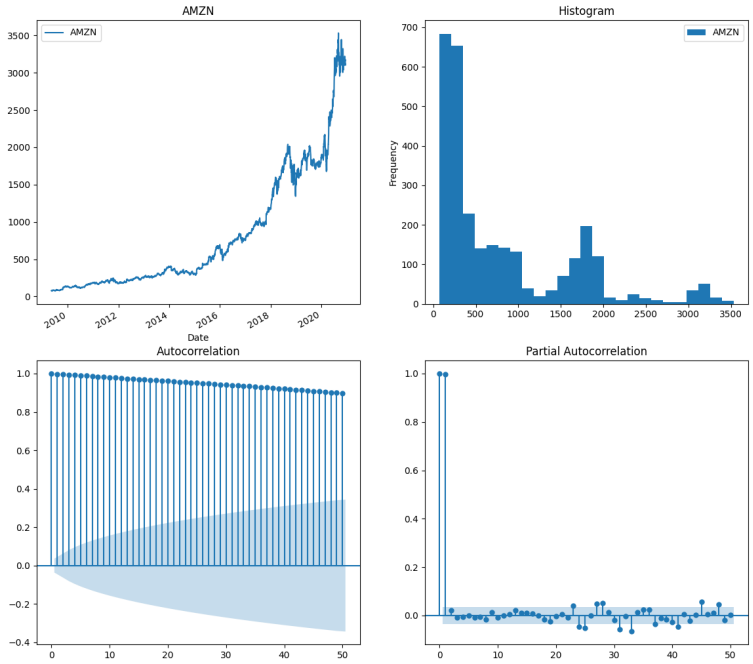
$$ACF_x(s,t) = cov(x_s, x_t) = E[(x_s - \mu_s)(x_t - \mu_t)]$$

$$PACF_x(s,t) = \frac{cov(x_s, x_{s-t}|x_{s-1}, ..., x_{s-t+1})}{\sqrt{var(x_s|x_{s-1}, ..., x_{s-t+1}) var(x_{s-t}|x_{s-1}, ..., x_{s-t+1})}}$$

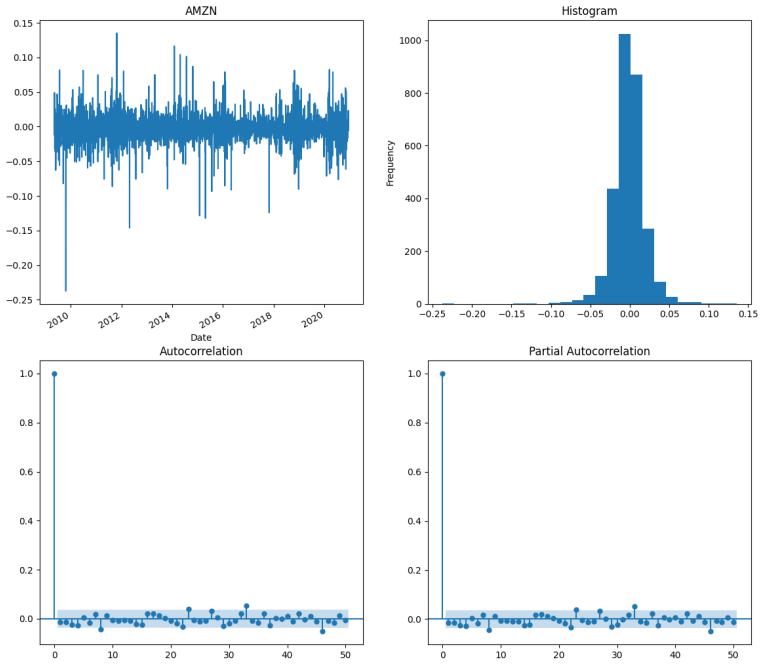
These tests will allow us to determine the conditional and non-conditional correlation of a given data point and higher order lags.

The initial results for the raw AMZN data for the ADF test, ACF, and PACF are below.

```
(Pre-Transformation)
ADF Statistic: -3.8446528690204484
p-value: 0.0024821130333634992
Critical Values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
```



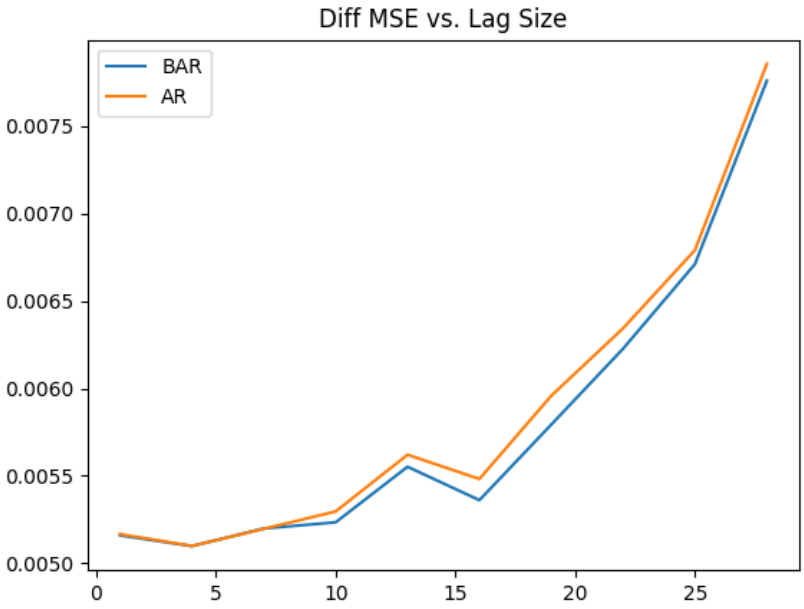
Here, we actually see that our time series passes the ADF test if we assume a p value tolerance of 0.05. Although, we also see a strong correlation with the first conditionally lagged variable and and see that the autocorrelation of the series decreases rather slowly. From the plotted series, it is apparent that the series has a significant trend but no seasonality. Given all this information, we can use first differencing to remove the trend. The exact mathematically operation is $y_t = \log(\frac{y_t}{y_{t-1}})$. The updated summary plots are below. We can see that the autocorrelation both conditional and non-conditional are significantly decreased. Thus, it will be much easier to model the series as a linear combination of past value given our assumptions.



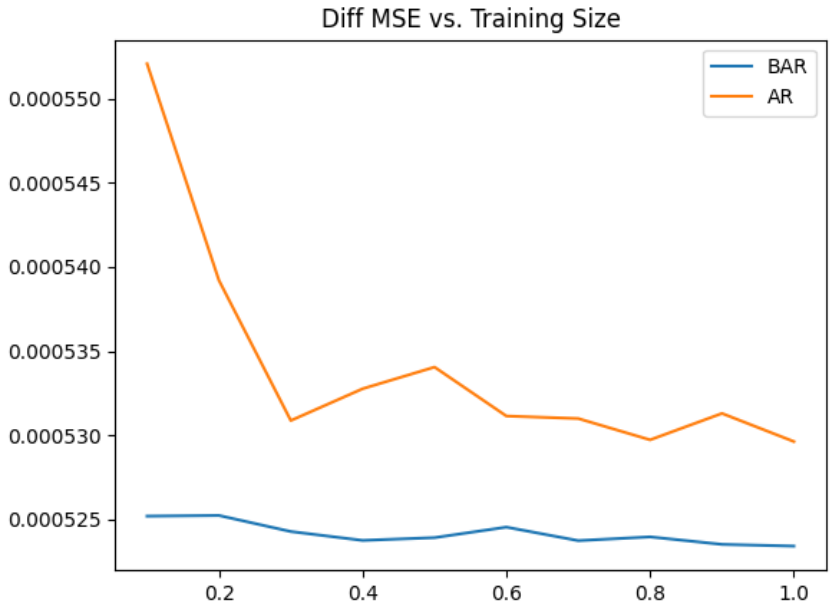

```
ADF Statistic: -54.74871295600178
p-value: 0.0
Critical Values:
        1%: -3.433
        5%: -2.863
        10%: -2.567
```

Lag & Training Size Tests

For the following, I used 30 averaged trials (given random row sampling) to compute the respective mean squared error value on the differenced data. From the partial autocorrelation analysis above, we know that the model is likely to perform best when using a single lagged variable. The following graph justifies this belief. We see that as we increase the lag used in the model the error increases. Something that is interesting to note is that the BAR model performs better than the autoregressive model as lags increase.



The following is the mean squared error of the bayesian and autoregressive model on various training sizes of data which were sampled randomly. It is apparent that the bayesian model performs quite well across all training sizes while the autoregressive model improves drastically as the training percentage increases but does not quite converge to its bayesian counterpart. This is likely due to the mean centered prior which helps bring the predictions towards the data's first moment.



Post Discussion

Initially, I wanted to explore the multivariate VAR and BSTS model with more complex priors but found myself gravitating towards the autoregressive model when I had issues implementing high dimensional matrix operations. This was a major bottleneck for me. Despite, I believe I got a solid introduction to time series analysis and many of the tips and tricks used for exploring relevant data. If I were to redo the project, I would definitely have spent my time wiser and pursued a more directed goal rather than trying to implement multiple models simultaneously.

Code Archive

```python
class TimeSeries:

    def __init__(self, data, date_col_name, col_name):
        self.series = data
        self.col_name = col_name

    def summaryPlots(self, lags=10, save_loc=''):
        _, layout = plt.figure(figsize=(14, 14)), (2, 2)
        ts_ax, h_ax = plt.subplot2grid(layout, (0, 0)), plt.subplot2grid(layout, (0, 1))
        ac_ax, pac_ax = plt.subplot2grid(layout, (1, 0)), plt.subplot2grid(layout, (1, 1))

        self.series.plot(ax=ts_ax)
        ts_ax.set_title(self.col_name)
        self.series.plot(ax=h_ax, kind='hist', bins=25)
        h_ax.set_title('Histogram')
        plot_acf(self.series, lags=lags, ax=ac_ax)
        plot_pacf(self.series, lags=lags, ax=pac_ax)

        plt.savefig(save_loc)

    def adfTest(self):
        result = adfuller(self.series)
        print('ADF Statistic: {}'.format(result[0]))
        print('p-value: {}'.format(result[1]))
        print('Critical Values:')
        for key, value in result[4].items():
            print('\t%s: %.3f' % (key, value))

    def to_frame(self):
        return pd.DataFrame(self.series)
```

```python
class AR:

    def __init__(self, data):
        self.data = data
        self.w = None

    def fit(self, lags, trainPct, holdoutPct):

        n, m = self.data.shape[0], self.data.shape[1]

        split_idx = int(n * (1-holdoutPct))


        X = pd.DataFrame(np.ones(n))
        X.set_index(self.data.index, inplace=True)
        for i in range(lags):
            X[i+1] = self.data.shift(i+1)
        X = X.iloc[lags:-lags]

        Y = self.data.iloc[lags:-lags]

        sample = np.random.choice(range(split_idx), size=int(split_idx*trainPct), replace=False)

        trainX, testX = X.iloc[sample].to_numpy(), X[split_idx:].to_numpy()
        trainY, testY = Y.iloc[sample].to_numpy(), Y[split_idx:].to_numpy()

        self.w = np.linalg.inv(trainX.T.dot(trainX)).dot(trainX.T.dot(trainY))

        pred = testX.dot(self.w)
        mse = np.sum((pred - testY) ** 2) / len(testY)

        return mse
```

```python
class BAR:

    def __init__(self, data):
        self.data = data
        self.w = None

    def fit(self, lags, trainPct, holdoutPct, lambdaParam=None):
        alpha, beta = random.randint(1,10), random.randint(1, 10)

        n, _ = self.data.shape[0], self.data.shape[1]

        split_idx = int(n * (1-holdoutPct))

        X = pd.DataFrame(np.ones(n))
        X.set_index(self.data.index, inplace=True)
        for i in range(lags):
            X[i+1] = self.data.shift(i+1)
        X = X.iloc[lags:-lags]

        Y = self.data.iloc[lags:-lags]

        if trainPct < 1:
            sample = np.random.choice(range(split_idx), size=int(split_idx*trainPct), replace=False)
            trainX, trainY = X.iloc[sample].to_numpy(), Y.iloc[sample].to_numpy()
        else:
            trainX, trainY = X[:split_idx].to_numpy(), Y[:split_idx].to_numpy()


        testX, testY = X[split_idx:].to_numpy(), Y[split_idx:].to_numpy()

        if lambdaParam == None:
            aErr, bErr = math.inf, math.inf
            while aErr > 10 ** -5 and bErr > 10 ** -5:

                eigVals, _ = np.linalg.eig(beta * (trainX.T.dot(trainX)))

                gamma = np.sum(eigVals / (eigVals + alpha))

                SN = np.linalg.inv((alpha * np.identity(trainX.shape[1])) + (beta * (trainX.T.dot(trainX))))
                mN = beta * (SN.dot(trainX.T).dot(trainY))

                alpha0 = gamma / mN.T.dot(mN)
                beta0 =  ((1 / (len(trainX) - gamma)) * (np.sum((trainY - trainX.dot(mN)) ** 2))) ** -1

                aErr, bErr = abs(alpha0 - alpha), abs(beta0 - beta)
                alpha, beta = alpha0, beta0

        lambdaParam = alpha / beta
        w = np.linalg.inv((lambdaParam * np.identity(trainX.shape[1])) + trainX.T.dot(trainX)).dot(trainX.T.dot(trainY))

        mse = np.sum((testX.dot(w) - testY) ** 2) / len(testY)

        return mse
```