Challenge

Needs to have a data science slant.

Market needs a benchmark. Today it is BTC. But that may change.

Determination of the index to better reflect the market is hard!

Define currencies to be included.

What are the type of currencies today?

Create index and evaluate it against major currencies over time.

Is the index less volatile?

Does it reflect the market?

Benchmark against sample portfolios; i.e., alpha

A leading indicator gives you an idea of what the final result will be, while the lagging indicator is that final result.

How does new indicator compare with S&P500, DJIA, Russell 2000, etc.?

Don’t we want to weight considering how much they have in the market now? How do we know when they release more? Do we penalize because they have more premined? When do we know other currencies released more coins?

Recommendation for algorithm?

Methodology to identify currencies?

Index EPS (earnings per share)

Index PE (price-earnings)

Index (book value per share, price-to-book)

Risk control indices

Exponentially-weighted volatility

Simple-weighted volatility

Domestic currency return index?

Number of exchanges?

Number of shares?

Trading volume?

Price change?

<http://www.nytimes.com/2003/08/31/business/strategies-4-ways-to-build-an-index-none-of-them-ideal.html>

ANOVA to determine if any factor really influences the dependent variable

Using change as opposed to absolute values could deal with autocorrelation. Pattern residuals can be dealt with by weighted regression or change the form of the dependent variables, for example use log.

R

Zoo for intraday (POSIXct or chron time stamps), day, month, year time series data

Filter() – moving average, linear filter

Rmetrics

fGarch

250 = days trading in a year

100 = turn into percent

calculate stock volatility in %

library(tseries)

data <- get.hist.quote('VOD.L')

price <- data$Close

ret <- log(lag(price)) - log(price)

vol <- sd(ret) \* sqrt(250) \* 100

multivariate time series analysis

machine learning with time series

multi-step forecast

multivariate time series

<https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/>

**multivariate forecasting**

Time series forecasting is an important area of machine learning that is often neglected.

It is important because there are so many prediction problems that involve a time component. These problems are neglected because it is this time component that makes time series problems more difficult to handle.

You can’t just fire a machine learning algorithm at a time series dataset.

Time series data must be transformed into a supervised learning problem.

Time series data has temporal structure like trends and seasonality that must be handled.

Time series data has a forecast horizon.

There are a few conceptual steps you must make before you can start developing forecasting models.

There are also specialized terminology and algorithms to consider and use when working with time series data.

from pandas import DataFrame

from pandas import concat

def series\_to\_supervised(data, n\_in=1, n\_out=1, dropnan=True):

"""

Frame a time series as a supervised learning dataset.

Arguments:

data: Sequence of observations as a list or NumPy array.

n\_in: Number of lag observations as input (X).

n\_out: Number of observations as output (y).

dropnan: Boolean whether or not to drop rows with NaN values.

Returns:

Pandas DataFrame of series framed for supervised learning.

"""

n\_vars = 1 if type(data) is list else data.shape[1]

df = DataFrame(data)

cols, names = list(), list()

# input sequence (t-n, ... t-1)

for i in range(n\_in, 0, -1):

cols.append(df.shift(i))

names += [('var%d(t-%d)' % (j+1, i)) for j in range(n\_vars)]

# forecast sequence (t, t+1, ... t+n)

for i in range(0, n\_out):

cols.append(df.shift(-i))

if i == 0:

names += [('var%d(t)' % (j+1)) for j in range(n\_vars)]

else:

names += [('var%d(t+%d)' % (j+1, i)) for j in range(n\_vars)]

# put it all together

agg = concat(cols, axis=1)

agg.columns = names

# drop rows with NaN values

if dropnan:

agg.dropna(inplace=True)

return agg

raw = DataFrame()

raw['ob1'] = [x for x in range(10)]

raw['ob2'] = [x for x in range(50, 60)]

values = raw.values

data = series\_to\_supervised(values)

print(data)

ARMAV (autoregressive moving average vector) or ARAM process

<https://github.com/mhamilton723/tseries/blob/master/Time%20Series%20Estimator%20Demo.ipynb>

<https://machinelearningmastery.com/multi-step-time-series-forecasting/>

solutions

1. Direct multi-step forecast where you develop a prediction model for each time step. Because separate models are used, there is not opportunity to model the dependence between predictions.
2. Recursive multi-step forecast where you use a one-step model multiple times where prediction for the prior time step is used as input for making a prediction on the following time step. Because predictions are used in place of observations, this strategy allows prediction errors to accumulate such that performance can quickly degrade as the prediction time horizon increases. (what we say with mhamilton’s approach)
3. Direct-recursive hybrid strategies uses #1 and #2 where a separate model can be constructed for each time step to predict, but each model may use predictions made by prior models. This combo method overcomes limitation of each.
4. Multiple output strategy involves developing a model capable of predicting the entire forecast sequence in one-shot manner. These are more complex as they can learn dependence structure between inputs and outputs as well as between outputs. They could be slower to train and require more data to avoid overfitting.

<https://machinelearningmastery.com/make-sample-forecasts-arima-python/>

<https://thuijskens.github.io/2016/08/03/time-series-forecasting/>

The difference with my previous encounters with time series analyses was that now I had to provide longer term forecasts (which in itself is an ambiguous term, as it depends on the context) for a large number of time series (~500K). This prevented me from using some of the classical methods mentioned before, because

1. classical ARIMA models are typically well-suited for short-term forecasts, but not for longer term forecasts due to the convergence of the autoregressive part of the model to the mean of the time series; and
2. the MCMC sampling algorithms for some of the Bayesian state-space models can be computationally heavy. Since I needed forecasts for a lot of time series quickly this ruled out these type of algorithms.

Instead, I opted for a more algorithmic point of view, as opposed to a statistical one, and decided to try out some machine learning methods. However, most of these methods are designed for independent and identically distributed (IID) data, so it is interesting to see how we can apply these models to non-IID time series data.

Forecasting strategies

Throughout this post we will make the following non-linear autoregressive representation (NAR) assumption. Let yt

denote the value of the time series at time point t

, then we assume that

yt+1=f(yt,…,yt−n+1)+ϵt,

for some autoregressive order n

and where ϵt

represents some noise at time t

and f

is an arbitrary and unknown function. The goal is to learn this function f

from the data and obtain forecasts for t+h

, where h∈{1,…,H}

. Hence, we are interested in predicting the next H

data points, not just the H

-th data point, given the history of the time series.

When H=1

(one-step ahead forecasting), it is straightforward to apply most machine learning methods on your data. In the case where we want to predict multiple time periods ahead (H>1

) things become a little more interesting.

In this case there are three common ways of forecasting:

iterated one-step ahead forecasting;

direct H

-step ahead forecasting; and

multiple input multiple output models.

The main advantages of the MIMO forecasting strategy are that

only one model is trained instead of H

different models;

no conditional independence assumptions are made (c.f. direct strategy); and

there is no accumulation of error of individual forecasts (c.f. iterated strategy).

One constraint of the MIMO strategy is that all horizons H

are to be forecasted with the same model, which limits our flexibility. One approach to combat this assumption is to combine the direct and MIMO strategy, and is called the DIRMO strategy2. The general idea is to split the forecasting horizon H

into m=Hb

blocks of length b

(where b∈{1,…,H}

). We then train m

different models, where each model is used to predict one of the blocks in a MIMO fashion.

If your ultimate goal is more explanatory rather than predictive in nature, you may find that more classical models like state-space models will give you better bang for your buck. Bayesian dynamic linear models (DLMs) in particular work nicely here, because of their flexibility and ease of interpretation (check out this post over at Stitch Fix for an excellent discussion of these models).

<http://multithreaded.stitchfix.com/blog/2016/04/21/forget-arima/>

you rarely have sufficient historical data to estimate these components with good precision. And, to make matters worse, validation is more difficult for time series models than it is for classifiers and your audience may not be comfortable with the embedded uncertainty. You need business acumen, luck, and Bayesian structural time series models. In my opinion, these models are more transparent than ARIMA – which still tends to be the go-to method. So, how does one navigate such treacherous waters? They also facilitate better handling of uncertainty, a key feature when planning for the future.

This model predicts the holdout period quite well as measured by the MAPE (mean absolute percentage error). However, the model does not tell us much about the time series itself. In other words, we cannot visualize the “story” of the model. All we know is that we can fit the data well using a combination of moving averages and lagged terms.

A different approach would be to use a Bayesian structural time series model with unobserved components. This technique is more transparent than ARIMA models and deals with uncertainty in a more elegant manner. It is more transparent because its representation does not rely on differencing, lags and moving averages. You can visually inspect the underlying components of the model. It handles uncertainty in a better way because you can quantify the posterior uncertainty of the individual components, control the variance of the components, and impose prior beliefs on the model. Last, but not least, any ARIMA model can be recast as a structural model.

<https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3>

<https://facebook.github.io/prophet/>

<https://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/>

is our time series linear or non-linear?? How can you tell?

Must perform cointegration test to see if two series are tied together.

Major challenge with forecasting is that it fails to give good predictions on a long horizon.

Machine learning approach

* use features
* break out seasonality if it has it by adding those features (date)
* can introduce cross sectional data
* can introduce lags as features
* can transform features (polynomials)
* financial factors
* use feature selection

maybe use ARIMA and machine learning prediction to compare results, which one is best??