Uni-variate multi-step time series forecasting

November 9, 2022

1 Introduction

1. **Forecasting task:** Multi-step forecasting of river water level data .
2. **Data sets involved**
   - Water level time series from a water level sensor node
     - Water level daily means in m; sampling interval is $5min$ hence we calculate the daily mean $W = \frac{\sum_{i=1}^{288} w_i}{288}$
     - Time spans: From 11th Feb 2021 to 1st Dec 2021
     - Resolution: Daily hence,294 data points/days
     - Normalized water level figures so that the maximum value for each column is 1

Modeling

Prediction is a subset of the estimation problem. Other subsets are smoothing and filtering. The original prediction problem was stated as - Given information upto a certain point in time, make a intelligent guess of the next instance. In multi-step prediction the number of instances to be predicted is usually more than one $(t-1, t-2, t-3, ..., t-n)$. In practice, the error margin is expected to increase with increase in time steps(to be highlighted). According to the decomposition property, given any 2 random variables, x and y can be decomposed into,

$$y = \mathbb{E}(y|x) + \epsilon$$

The expectation of the error ϵ given x, $\mathbb{E}(\epsilon|x) = 0$ is zero, hence ϵ is not related to x. This property is used in fine-tuning a model and informing how much of y can predicted using x. Therefore, the best predictor is the conditional expectation where a variable can be decomposed into the conditional expectation and the orthogonal error term.

Case specific

Predictive modeling can be described as the mathematical problem of approximating a mapping function f from inputs x to output variables y.

$$f \approx (x, y)$$

$$y \approx f(x)$$

$$\hat{y} = f(x) + \epsilon; prediction$$

The x and y are usually in matrix form. The $e \approx \mathcal{N}(\mathcal{O}, \sigma)$ is the prediction error. in our case the data is in a time series form hence we need to perform some pre-processing operations that will prepare the data for the model.

Definitions

- **Time step:** In this case one time step is equal to one day, be it a historical time step or a future time step.
- **Window:** number of historical time steps $(t-1, t-2, t-3, ..., t-(n-1))$ that are being considered in a particular forecasting instance.
- **Prediction Horizon:** Number of future time steps $(t+1, t+2, t+3, t+4, ..., t+(n))$ that are being considered in a particular forecasting instance. The prediction horizon in these prediction task will be set at five future time steps. Clearly ,
  - The predictions requires knowledge of the past/present in addition to the model.
  - The quality of the prediction is dependent on the quality of the model, prediction horizon and the uncertainty level(error — variance) - $y(t) = x(t) + v(t)$

Data preparation

Train-Test preparation

In the section there are 2 important parameters, the length of the window *hstep* and the length of the horizon *fstep*. The *fstep* is fixed and the *hstep* is varied to find the optimal window size. The layout below shows how the x-matrix and the y-matrix are created from the one-dimension time series.

$$\begin{bmatrix} 0.27592914 \\ 0.24092476 \\ 0.46736996 \\ 0.57636297 \\ 0.72730505 \\ 0.506776 \\ 0.34338928 \\ 0.30609814 \\ 0.26900873 \\ 0.24930302 \\ 0.28211601 \\ 0.34060019 \\ 0.33476536 \\ 0.2779396 \\ 0.24109471 \\ 0.23227272 \\ 0.23416678 \\ 0.23416678 \\ 0.20879137 \\ 0.28878034 \\ \Downarrow \end{bmatrix} \Rightarrow \begin{bmatrix} 0.27592914 & 0.24092476 & 0.46736996 & 0.57636297 \\ 0.24092476 & 0.46736996 & 0.57636297 & 0.72730505 \\ 0.46736996 & 0.57636297 & 0.72730505 & 0.506776 \\ 0.57636297 & 0.72730505 & 0.506776 & 0.34338928 \\ 0.72730505 & 0.506776 & 0.34338928 & 0.30609814 \\ 0.506776 & 0.34338928 & 0.30609814 & 0.26900873 \\ 0.34338928 & 0.30609814 & 0.26900873 & 0.24930302 \\ 0.30609814 & 0.26900873 & 0.24930302 & 0.28211601 \\ \Downarrow & & & \end{bmatrix} \Rightarrow \begin{bmatrix} 0.72730505 & 0.506776 & 0.34338928 \\ 0.506776 & 0.34338928 & 0.30609814 \\ 0.34338928 & 0.30609814 & 0.26900873 \\ 0.30609814 & 0.26900873 & 0.24930302 \\ 0.26900873 & 0.24930302 & 0.28211601 \\ 0.24930302 & 0.28211601 & 0.34060019 \\ 0.28211601 & 0.34060019 & 0.33476536 \\ 0.34060019 & 0.33476536 & 0.2779396 \\ 0.33476536 & 0.2779396 & 0.24109471 \\ 0.2779396 & 0.24109471 & 0.23227272 \\ \Downarrow & & \end{bmatrix}$$

$$\begin{bmatrix} timeseries \\ [294,] \end{bmatrix} \Rightarrow \begin{bmatrix} xtrain - matrix \\ [197, 4] \end{bmatrix} \Rightarrow \begin{bmatrix} ytrain - matrix \\ [197, 2] \end{bmatrix}$$

$$\begin{bmatrix} xtest - matrix \\ [81, 4] \end{bmatrix} \Rightarrow \begin{bmatrix} ytest - matrix \\ [81, 2] \end{bmatrix}$$

Listing 1: Function utilized in generating the train and test sets shown above

```
#train_test_split
main_series =df[df.columns[0]].to_numpy()
data_size = int(main_series.shape[0] * .70)
data_train = main_series[:data_size].flatten()
data_test = main_series[data_size:].flatten()
hstep = 4
fstep = 2
def train_test_data_split(dataset, hstep, fstep):
    x, y = [], []
    for i in range(h_step, len(dataset)- hstep):
        x.append(dataset[i - hstep:i])
        y.append(dataset[i : i+fstep])
    return np.array(x),np.array(y)
x_train, y_train = train_test_data_split(data_train, hstep , fstep)
x_test, y_test = train_test_data_split(data_test, hstep , fstep )
```

Data reshape and LSTM model fitting

To facilitate fitting, the inputs are reshaped.

epochs describing this

Listing 2: Input reshape

```
n_features = 1
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], n_features))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], n_features))
```

Listing 3: LSTM Model

```
# define model
model = Sequential()
model.add(Bidirectional(LSTM(200, activation='relu'), input_shape=(x_train.shape[1], n_features)))
model.add(Dense(y_test.shape[1]))
model.compile(optimizer='adam', loss='mse')
# fit model
history = model.fit(x_train, y_train, epochs=100, batch_size= 20, validation_data=(x_test, y_test)
y_pred = model.predict(x_test)
y_pred.shape
```

Model Evaluation

The function *evaluate − model*, evaluates the model's performance for the overall prediction instance and for the specific horizon time steps. The Root Mean Squared Error (RMSE) is the metric adopted in this work. Compared to the Mean Absolute Error(MAE, it is more sensitive.

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2)}$$

How does this change depending on how many steps ahead?

Listing 4: Model evaluation

```
def evaluate_model(y_test , y_pred):
    scores = []
    #scores for each day
    for i in range (y_test.shape[1]):
        mse = mean_squared_error(y_test[:,i], y_pred[:,i])
        rmse = np.sqrt(mse)
```

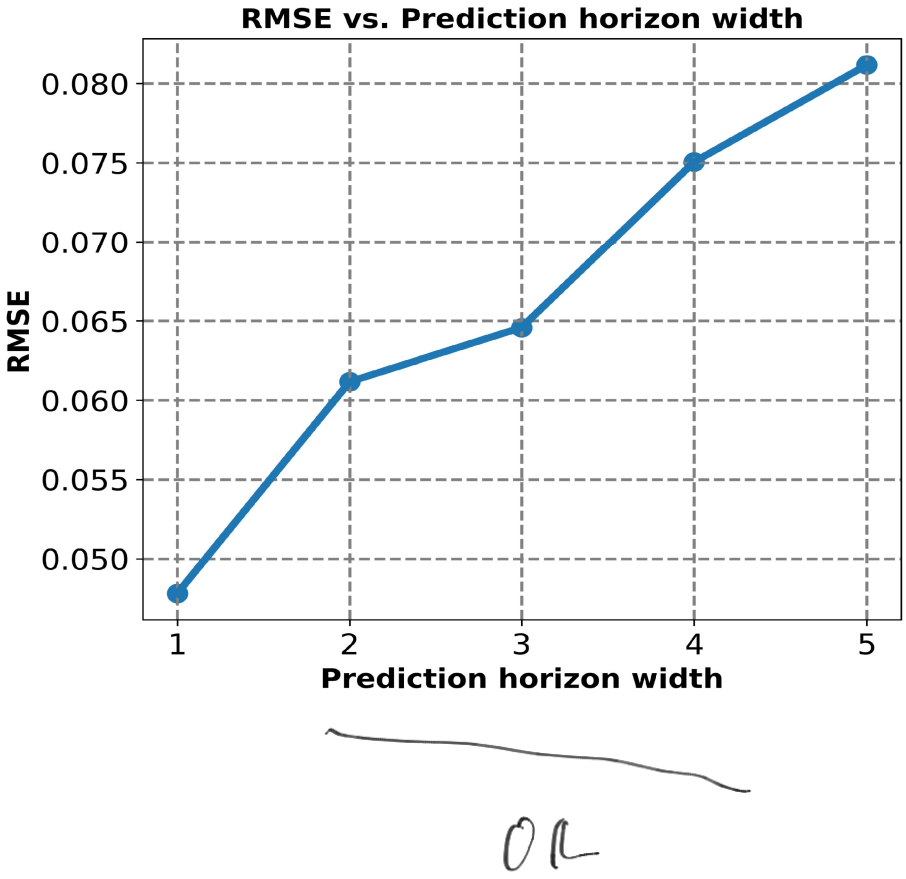
```
        scores.append(rmse)
    #scores for the whole prediction exercise
    overall_score = 0
    for row in range (y_test.shape[0]):
        for col in range (y_pred.shape[1]):
            overall_score = overall_score + (y_test[row,col]-y_pred[row,col])**2
    overall_score = np.sqrt(overall_score/(y_test.shape[0] * y_pred.shape[1]))
    return overall_score , scores
```

Results

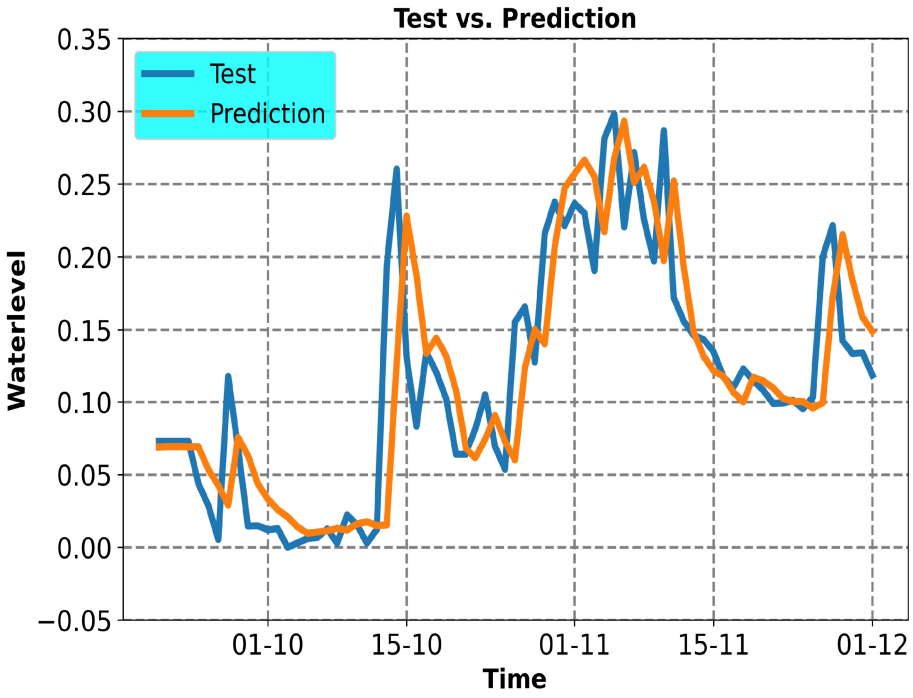
Example: window $hstep = 8$, horizon $fstep = 5$

Listing 5: RMSE results

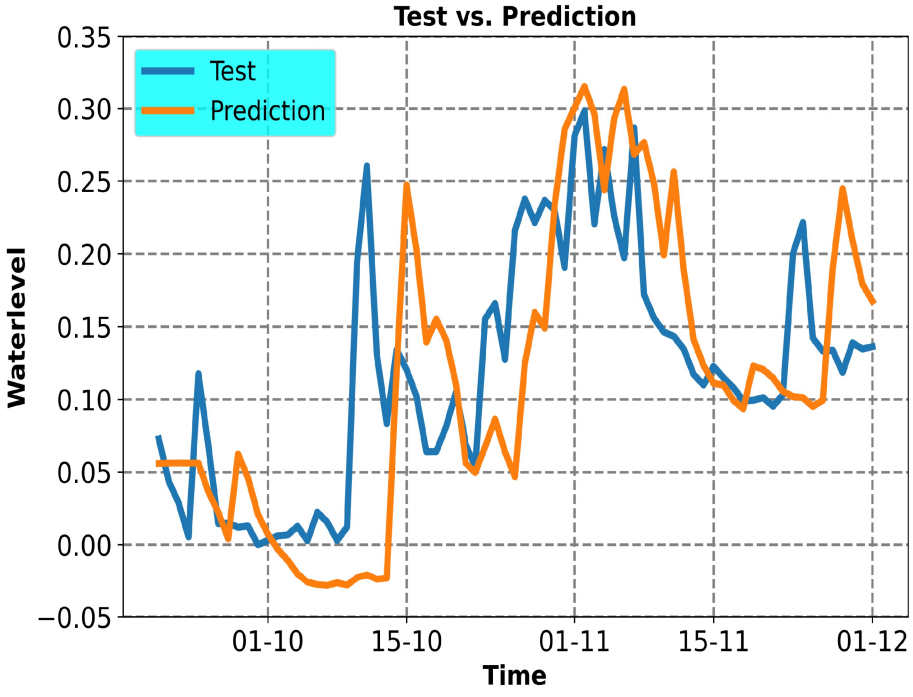
```
[0.047822802790704104,
 0.06116031862176686,
 0.06457476680787931,
 0.0750577108284362,
 0.08116707180316897]
```



Prediction vs Test at fstep = 1 , $RMSE = 0.0478$



Prediction vs Test at fstep = 4, $RMSE = 0.075$



Next step

Evaluating the optimal *hstep*