Predicting Milk Production via RNN

May 1, 2024

0.1 The Data

** Source: https://datamarket.com/data/set/22ox/monthly-milk-production-pounds-per-cow-jan-62-dec-75#!ds=22ox&display=line **

Monthly milk production: pounds per cow. Jan 62 - Dec 75

** Import numpy pandas and matplotlib **

** Use pandas to read the csv of the monthly-milk-production.csv file and set index_col='Month' **

** Check out the head of the dataframe**

[10]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")

# Load data
file_path = '/content/monthly-milk-production.csv'
df = pd.read_csv(file_path, index_col='Month', parse_dates=True)

# Check the data
print(df.head())
```

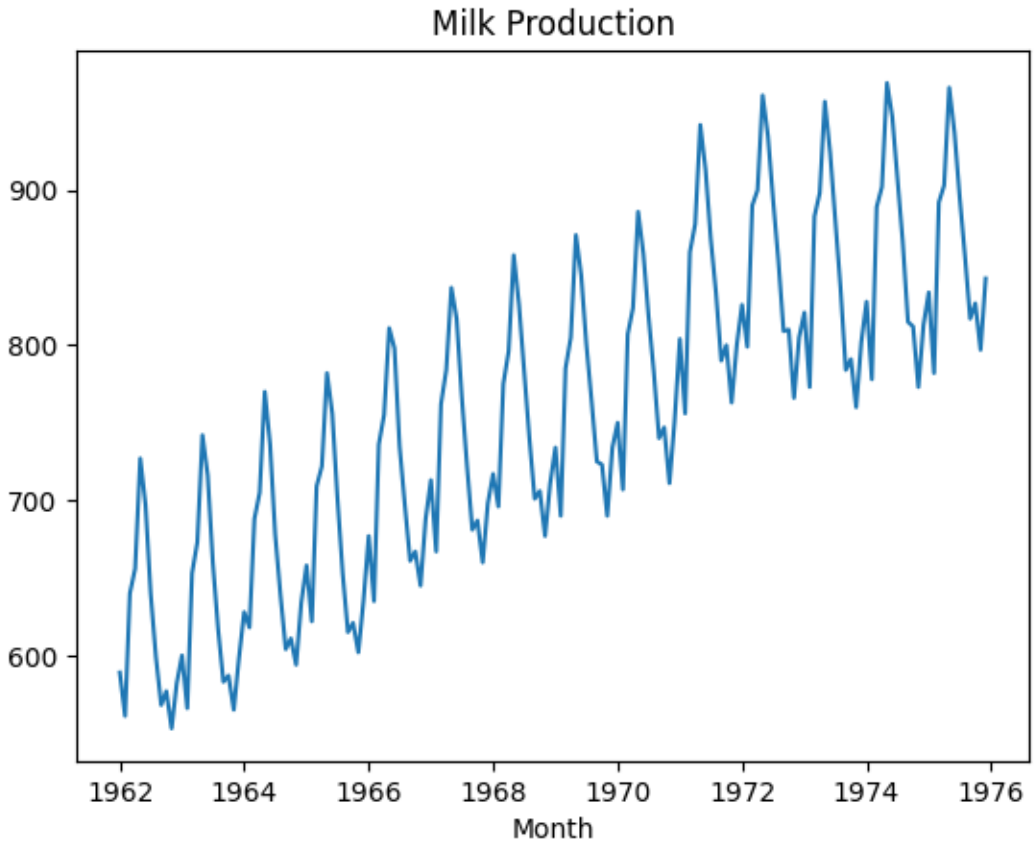
```
                     Milk Production
Month
1962-01-01 01:00:00            589.0
1962-02-01 01:00:00            561.0
1962-03-01 01:00:00            640.0
1962-04-01 01:00:00            656.0
1962-05-01 01:00:00            727.0
```

** Plot out the time series data. **

[11]:
```
import matplotlib.pyplot as plt

plt.plot(df)
plt.title('Milk Production')
```

```
plt.xlabel('Month')
plt.show()
```



0.1.1 Train Test Split

** Let's attempt to predict a year's worth of data. (12 months or 12 steps into the future) **

[12]:
```
df_train = df.iloc[:156]
df_test = df.iloc[156:]
```

0.1.2 Scale the Data

** Use sklearn.preprocessing to scale the data using the MinMaxScaler. Remember to only fit_transform on the training data, then transform the test data. You shouldn't fit on the test data as well, otherwise you are assuming you would know about future behavior!**

[13]:
```
from sklearn.preprocessing import MinMaxScaler

data_train = df_train.values.astype(np.float32).reshape(-1, 1)
data_test = df_test.values.astype(np.float32).reshape(-1, 1)
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
data_tr_normalized = scaler.fit_transform(data_train)
data_te_normalized = scaler.transform(data_test)
```

0.1.3 Batch Creation

** Create batches of length 12 that will be used to train your RNN model **

[14]:
```
sequence_length = 12
batch_size = 1

dataset = tf.keras.preprocessing.timeseries_dataset_from_array(
    data=data_tr_normalized[:-1],
    targets=data_tr_normalized[1:],
    sequence_length=sequence_length,
    batch_size=batch_size,
    shuffle=False
)
```

0.1.4 Model Creation

** Create a RNN model with two layers, each having 50 Neurons **

[15]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=(sequence_length, 1)),
    tf.keras.layers.SimpleRNN(50, return_sequences=True),
    tf.keras.layers.SimpleRNN(50),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')
```

0.1.5 Fitting the model

** Fit this model to your training set, using 50 epochs **

[16]:
```
epochs = 10
model.fit(dataset, epochs=epochs, verbose=1)
```

```
Epoch 1/10
144/144 [==============================] - 4s 8ms/step - loss: 0.0657
Epoch 2/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0199
Epoch 3/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0050
Epoch 4/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0066
Epoch 5/10
```

```
144/144 [==============================] - 1s 6ms/step - loss: 0.0069
Epoch 6/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0066
Epoch 7/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0111
Epoch 8/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0104
Epoch 9/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0171
Epoch 10/10
144/144 [==============================] - 1s 6ms/step - loss: 0.0120
```

[16]:
```
<keras.src.callbacks.History at 0x7acf1c044520>
```

0.1.6 Predicting milk production

** Now determine your model's prediction on the test data (last 12 months of the dataset) **

[17]:
```
last_train_sequence = data_tr_normalized[-sequence_length:]
last_train_sequence = last_train_sequence.reshape(1, sequence_length, 1)

predicted_future = []
current_sequence = last_train_sequence

for _ in range(12):
    next_step = model.predict(current_sequence)
    predicted_future.append(next_step.ravel()[0])

    current_sequence = np.roll(current_sequence, -1, axis=1)
    current_sequence[0, -1, 0] = next_step
```

```
1/1 [==============================] - 0s 284ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
```

0.1.7 Mean Squared Error

** Now determine your model's MSE on the test set **

[18]:
```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(data_te_normalized, np.array(predicted_future))
print("Mean Squared Error on test set:", mse)
```

```
Mean Squared Error on test set: 0.039555565
```