all_seasons_simplified_second_order

December 7, 2022

1 All Seasons - Simplified(long time frame - Second-Order Markov Chain)

1.1 Import libraries and dataset

[ ]:

```
import pandas as pd
import numpy as np
from datetime import datetime
date_format = "%Y-%m-%d"
```

[ ]:

```
all_seasons = pd.read_csv('Datasets/all_seasons.csv')
all_seasons = all_seasons[['datetime', 'conditions']]
```

[ ]:

```
all_seasons.head()
```

[ ]:

```
     datetime        conditions
0  2000-01-01  Partially cloudy
1  2000-01-02             Clear
2  2000-01-03             Clear
3  2000-01-04             Clear
4  2000-01-05             Clear
```

1.2 Classify and separate data

[ ]:

```
simplifier = {'Overcast':'no_rain', 'Partially cloudy':'no_rain', 'Clear':
 ↪'no_rain', 'Rain, Partially cloudy':'rain', 'Rain':'rain', 'Rain, Overcast':
 ↪'rain'}

all_seasons['condition'] = all_seasons['conditions'].map(simplifier)
```

[ ]:

```
all_seasons.head()
```

[ ]:

```
     datetime        conditions condition
0  2000-01-01  Partially cloudy   no_rain
1  2000-01-02             Clear   no_rain
2  2000-01-03             Clear   no_rain
3  2000-01-04             Clear   no_rain
```

```
4  2000-01-05             Clear   no_rain
```

[ ]:

```
all_seasons = all_seasons[['datetime', 'condition']]
```

[ ]:

```
all_seasons.head()
```

[ ]:

```
     datetime condition
0  2000-01-01   no_rain
1  2000-01-02   no_rain
2  2000-01-03   no_rain
3  2000-01-04   no_rain
4  2000-01-05   no_rain
```

[ ]:

```
train_start_date = '2002-01-01'
train_end_date = '2017-12-31'
all_seasons_train = all_seasons.loc[all_seasons['datetime'].
 ↪between(train_start_date, train_end_date)]
all_seasons_train = all_seasons_train.reset_index()

test_start_date = '2018-01-01'
test_end_date = '2021-12-31'
all_seasons_test = all_seasons.loc[all_seasons['datetime'].
 ↪between(test_start_date, test_end_date)]
all_seasons_test = all_seasons_test.reset_index()
```

1.3 Calculate proportions of conditions & Create transition matrix

[ ]:

```
# Initialize count variables

# 0: 'clear' - C
# 1: 'partially_cloudy' - PC
# 2: 'overcast' - OV
# 3: 'rain' - R
# 4: 'rain_partially_cloudy' - RPC
# 5: 'rain_overcast' - ROV

conditions = ['rain', 'no_rain']
prev_conditions = [f"{state_0}->{state_1}" for state_0 in conditions for␣
 ↪state_1 in conditions]
prev_conditions
```

[ ]:

```
['rain->rain', 'rain->no_rain', 'no_rain->rain', 'no_rain->no_rain']
```

[ ]:

```
# Adding a column to identify past two states

for i in range(2, len(all_seasons_train)):
    state_0 = all_seasons_train.loc[i-2, 'condition']
    state_1 = all_seasons_train.loc[i-1, 'condition']
    all_seasons_train.loc[i, 'prev_states'] = f"{state_0}->{state_1}"

all_seasons_train
```

[ ]:

```
      index    datetime condition       prev_states
0       731  2002-01-01   no_rain               NaN
1       732  2002-01-02      rain               NaN
2       733  2002-01-03      rain     no_rain->rain
3       734  2002-01-04   no_rain        rain->rain
4       735  2002-01-05   no_rain     rain->no_rain
…       …          …          …               …
5839   6570  2017-12-27   no_rain  no_rain->no_rain
5840   6571  2017-12-28   no_rain  no_rain->no_rain
5841   6572  2017-12-29   no_rain  no_rain->no_rain
5842   6573  2017-12-30   no_rain  no_rain->no_rain
5843   6574  2017-12-31   no_rain  no_rain->no_rain

[5844 rows x 4 columns]
```

[ ]:

```
# Creating a count matrix
# transition_counts = prev_conditions x conditions matrix

transition_counts = np.zeros((len(prev_conditions), len(conditions)))

for i in range(len(transition_counts)):
    for j in range(len(transition_counts[0])):
        transition_counts[i][j] = len(all_seasons_train[(all_seasons_train.
 ↪condition == conditions[j]) & (all_seasons_train.prev_states ==␣
 ↪prev_conditions[i])])

transition_counts
```

[ ]:

```
array([[ 209.,  208.],
       [  73.,  402.],
       [ 208.,  267.],
       [ 401., 4074.]])
```

[ ]:

```
# Turning count matrix into proportions by normalizing across rows

def normalize(arr):
    total = sum(arr)
    if total == 0:
        return arr
    return arr / total
```

```
transition_prob = np.apply_along_axis(normalize, 1, transition_counts)
transition_prob
```

```
[ ]: array([[0.50119904, 0.49880096],
            [0.15368421, 0.84631579],
            [0.43789474, 0.56210526],
            [0.08960894, 0.91039106]])
```

```
[ ]: # Verifying rows sum to 1
     np.apply_along_axis(sum, 1, transition_prob)
```

```
[ ]: array([1., 1., 1., 1.])
```

```
[ ]: all_seasons_test.head(2)
```

```
[ ]:    index    datetime  condition
     0   6575  2018-01-01    no_rain
     1   6576  2018-01-02    no_rain
```

First day of 2018: no_rain->no_rain

```
[ ]: tuple_to_row = {('rain','rain'):0, ('rain','no_rain'):1, ('no_rain','rain'):2,␣
      ↪('no_rain','no_rain'):3}
     tuple_to_row[('rain','rain')]
```

```
[ ]: 0
```

```
[ ]: def predict_weather_simplified(test_data):
         state = {0:'rain', 1:'no_rain'}
         prev_conditions = {0: 'rain->rain', 1:'rain->no_rain', 2:'no_rain->rain', 3:
      ↪'no_rain->no_rain'}
         tuple_to_row = {('rain','rain'):0, ('rain','no_rain'):1, ('no_rain','rain'):
      ↪2, ('no_rain','no_rain'):3}

         n = len(test_data) - 2 #how many steps to test
         start_state = (test_data.condition[0], test_data.condition[1])
         test_result = test_data.copy()

         prev_state = start_state
         result = [test_data.condition[0], test_data.condition[1]]
         while n-1:
             curr_state = np.random.choice([0,1],␣
      ↪p=transition_prob[tuple_to_row[prev_state]]) #taking the probability from␣
      ↪the transition matrix
             result.append(state[curr_state])
             prev_state = (prev_state[1], state[curr_state])
             n -= 1
```

```
        curr_state = np.random.choice([0,1],␣
 ↪p=transition_prob[tuple_to_row[prev_state]]) #taking the probability from␣
 ↪the transition matrix
        result.append(state[curr_state])

    test_result['predicted_condition'] = result

    return test_result

def find_accuracy(predicted_result):
    correct_count = 0.0

    for i in range(len(predicted_result)):
        if predicted_result.loc[i, 'condition'] == predicted_result.loc[i,␣
 ↪'predicted_condition']:
            correct_count += 1

    correct_prop = correct_count / len(predicted_result)

    return correct_prop

def run_predictions_return_avg_accuracy(test_data, trial_count):
    accuracy_sum = 0.0
    for i in range(trial_count):
        predicted_result = predict_weather_simplified(test_data)
        accuracy = find_accuracy(predicted_result)
        accuracy_sum += accuracy
    avg_accuracy = accuracy_sum / trial_count

    return avg_accuracy
```

[ ]:

```
# Sample prediction (for table graphic)

sample_prediction = predict_weather_simplified(all_seasons_test)
sample_accuracy = find_accuracy(sample_prediction)
print(sample_prediction.head())
print(sample_accuracy)
```

```
   index    datetime condition predicted_condition
0   6575  2018-01-01   no_rain             no_rain
1   6576  2018-01-02   no_rain             no_rain
2   6577  2018-01-03   no_rain                rain
3   6578  2018-01-04   no_rain             no_rain
4   6579  2018-01-05   no_rain             no_rain
0.7741273100616016
```

[ ]:

```
run_predictions_return_avg_accuracy(all_seasons_test, 100)
```

[ ]:

```
0.7696988364134153
```