Cadeias de Markov
UFRGS - FIS01082

Aluno: Henrique Alexandre Boneto - 288744

Junho 2020

Cadeias de Markov

Uma cadeia de markov é um processo estocástico em que a probabilidade de eventos futuros depende somente do evento mais atual. Ou seja, temos um espaço de estados:

$$S = \{X_0, X_1, X_2, ..., X_t\}$$

E assim, o estado futuro X_{t+1} dependerá somente do estado atual X_t.

Espaço de Estados

O espaço de estados são todos os estados possíveis de ocorrerem para as variáveis aleatórias do processo:

$$S = \{1, 2, ..., N\}$$

Assim, podemos formalizar seguinte propriedade:

$$P(X_{t+1} = e|X_t = e_t, X_{t-1} = e_{t-1}, ..., X_0 = e_0) = P(X_{t+1=e}|X_t = e_t)$$

Matriz de transição

A matriz de transição é uma matriz formada pelas probabilidades de transição entre todos os estados possíveis da cadeia de markov. Por exemplo, considerando que um processo tenha somente dois estados e_1 e e_2, teríamos a seguinte matriz P:

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}$$

Onde p_{11} representa a probabilidade de transição do estado e_1 para e_1, p_{12} representa a probabilidade de transição do estado e_1 para e_2, e assim por diante. A partir dela que podemos calcular o estado futuro dado o estado presente. De forma geral os elementos da matriz são dados por [2]:

$$p_{ij} = P(X_{t+1} = j|X_t = i)$$

Distribuição Estacionária

A distribuição estacionária é a distribuição que o sistema pode adquirir após o processo iterativo para grandes valores de t.

$$X_1 = X_0 P$$

$$X_2 = X_1 P = X_0 P^2$$

$$...$$

$$X_n = X_0 P^n$$

Se o limite $lim_{n \to \infty} X_0 P^n$ existir, teremos então o estado estacionario π.

Uma solução algébrica para tal problema é a seguinte:

$$\pi = e(I + E - P)^{-1}$$

Onde e é um vetor formado somente por uns, I é a matriz identidade é uma matriz formada por uns e P é a matriz de transição.

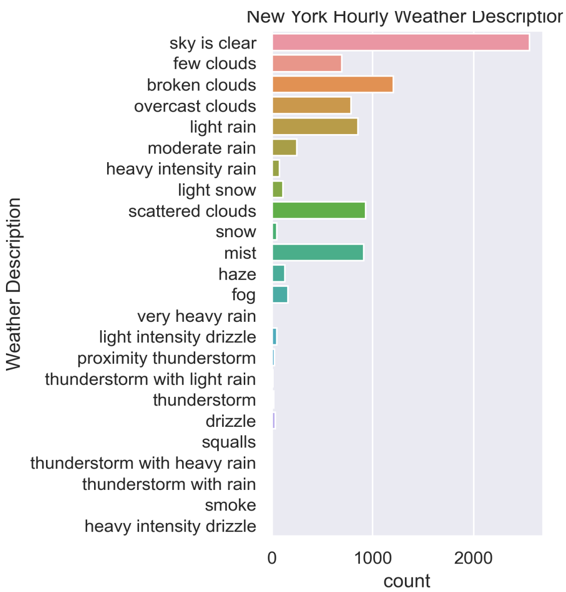
Série de Dados

A série de dados escolhida para realizar este trabalho foi encontrada no site Kaggle [1]. Ela é formada por dados históricos da descrição do tempo de várias cidades importantes, principalmente dos Estados Unidos. A cidade escolhida para análise foi a cidade de Nova York (New York) entre o período de 2015-01-01 e 2016-01-06. A série de dados final ficou com 8.881 linhas, formada por duas colunas 'datetime' contendo a data da descrição do tempo e 'weather_desc' contendo a descrição do tempo. Abaixo, pode-se ver melhor uma amostra dos dados:

	datetime	weather_desc
19716	2015-01-01 00:00:00	sky is clear
19717	2015-01-01 01:00:00	sky is clear
19718	2015-01-01 02:00:00	sky is clear
19719	2015-01-01 03:00:00	sky is clear
19720	2015-01-01 04:00:00	sky is clear
...	...	...
28592	2016-01-05 20:00:00	sky is clear
28593	2016-01-05 21:00:00	sky is clear
28594	2016-01-05 22:00:00	sky is clear
28595	2016-01-05 23:00:00	sky is clear
28596	2016-01-06 00:00:00	sky is clear

8881 rows × 2 columns

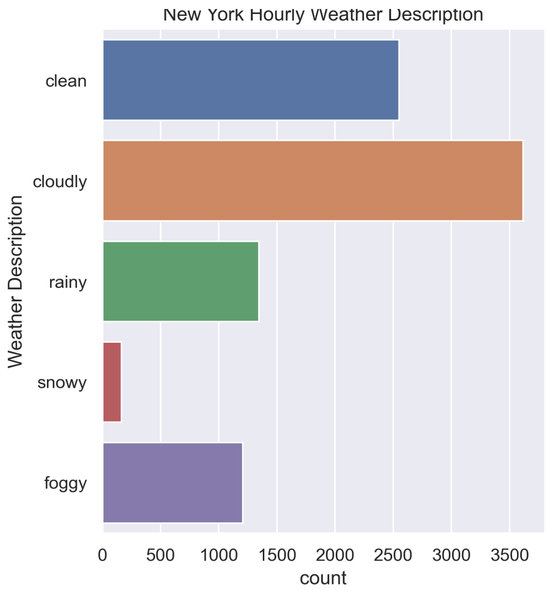
Para ter uma melhor ideia sobre as informações contidas nas descrições, fez-se uma contagem sobre todas os valores únicos contidos nos dados. Abaixo, a figura mostra que existem 24 descrições únicas durante o período:



Como várias das descrições contém poucas contagens e muitas delas fazem parte de uma mesmo grupo maior (por exemplo 'few clouds' e 'broken clouds' podiam fazer parte de simplesmente um grupo maior chamado 'cloudly') resolveu-se criar somente 5 descrições principais da seguinte maneira:

- Se a coluna 'weather_desc' contém a palavra 'clear', criar nova descrição 'clean'.
- Se a coluna 'weather_desc' contém as palavras 'clouds', criar nova descrição 'cloudly'.
- Se a coluna 'weather_desc' contém a palavra 'rain ou drizzle ou thunderstorm ou squalls', criar nova descrição 'rainy'.
- Se a coluna 'weather_desc' contém a palavra 'mist ou haze ou fog ou smoke', criar nova descrição 'foggy'.
- Se a coluna 'weather_desc' contém a palavra 'snow', criar nova descrição 'snowy'.

O resultado das novas contagens pode ser visto abaixo:



Para facilitar a manipulação dos dados, para cada uma das novas descrições, criou-se um índice indo de $0, 1, .., 4$ referentes respectivamente à ordem acima (clean $\rightarrow$ 0, cloudly $\rightarrow$ 1, ...). Isso facilitará na hora de definirmos as probabilidades de transição, por exemplo probabilidade de transição de cloudly para cloudly será chamada de p_{00}.

Após isso, foi feita a divisão dos dados em dados de treino, para calcular a matriz de probabilidade e dados de teste, para testar como o modelo irá se comportar.

- Para os dados de treino usou-se o período de '2015-01-01' a '2015-10-01'.
- Os dados de teste foram escolhidos em um dia inteiro, com 2 dias com diferentes tipos de tempo:
  - Dados de teste 1: escolhido entre o período de '2015-10-25' a '2015-10-26'. Estes dados contém todos os estados de tempo possíveis durante o dia.
  - Dados de teste 2: escolhido entre o período de '2016-01-05' a '2016-01-06'. Este dados contém somente as descrições 'clean' e 'cloudly' durante o dia.

Resultados

Cálculo da Matriz de Transição

Após a escolha e a manipulação dos dados foi feito o cálculo da matriz de transição de estados. A princípio, todos as transições são possíveis, totalizando 25 probabilidades de transição, já que temos 5 estados possíveis.

Para o cálculo, criou-se uma matriz P 5×5, e percorreu-se todos os valores de índice ordenados pela data. O índice de tempo mais atual foi acessado por $x[i]$ e chamado de j o estado futuro por $x[i+1]$ e chamado de k. Os índices dão acesso ao local onde deve-se calcular a probabilidade de transição da matriz (P[k][j]), por exemplo, se o estado futuro foi 'cloudly' (índice 1) e o mais atual foi 'clean', soma-se $1/N$ à P[1][0] (probabilidade do estado 1 dado que 0 ocorreu). Também é possível ver a função 'transition_matrix' feita em python na seção 'Implementação'.

O resultado da matriz foi aproximademente o seguinte:

$$P = \begin{pmatrix} 2.60e-1 & 4.92e-2 & 1.11e-2 & 1.98e-3 & 6.71e-3 \\ 5.02e-2 & 2.93e-1 & 1.95e-2 & 2.74e-3 & 1.17e-2 \\ 8.08e-3 & 2.28e-2 & 1.05e-1 & 3.05e-4 & 1.28e-2 \\ 1.67e-3 & 3.35e-3 & 0.00e+0 & 1.87e-2 & 1.52e-4 \\ 9.30e-3 & 8.54e-3 & 1.34e-2 & 1.52e-4 & 8.69e-2 \end{pmatrix}$$

É possível perceber que as maiores probabilidades estão nas diagonais, ou seja as probabilidades de que o estado continue o mesmo. Isso se deve ao tipo de dados que temos, pois imagina-se que em um único dia o tempo de uma cidade não varie muito de hora em hora, e sim demore algumas horas até estabelecer um novo tipo de tempo.

Previsões

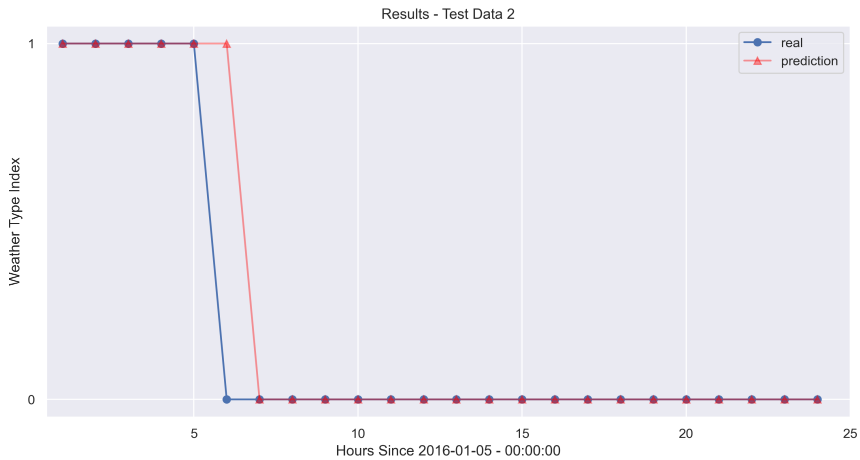
Para fazer as previsões foi implementada a função 'make_predictions'. Foi feito a varredura sobre os dados de teste e calculado X_{t+1}:

$$X_{t+1} = X_t P$$

Onde X_t é o valor atual do dado de teste e x_{t+1} retornará as probabilidades para cada um dos estados possíveis. Então escolhe-se o estado com maior probabilidade possível e compara-se com o real futuro contido nos dados de teste.

Abaixo podemos ver o resultado sobre os dados de teste 1, lembrando que foi escolhido um dia inteiro, ou seja 24 horas/linhas de dados:

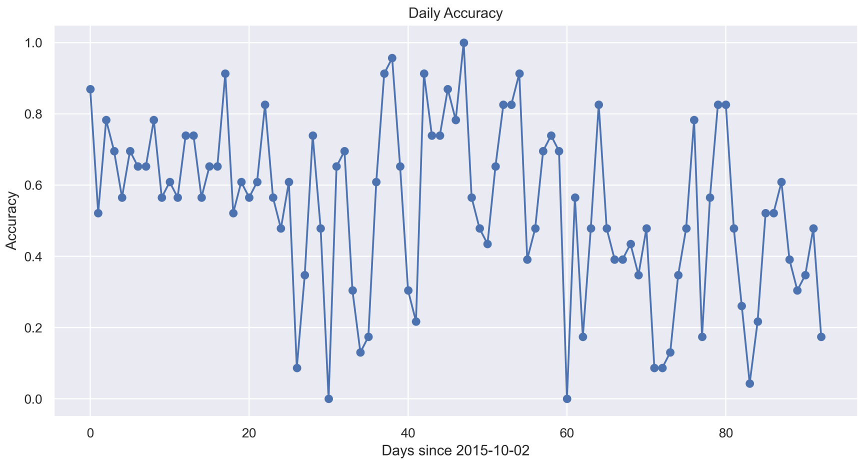
Agora para os dados de teste 2:



É possível perceber que por conta do modelo escolher sempre o estado com maior probabilidade, ele tende a ser bom para dias em que o tempo está estável, e 'se confunde' quando há trocas, entretanto adapta-se rapidamente a primeira vista.

Acurácia

Para ver melhor como o modelo performa, é interessante ver a acurácia sobre vários dias diferentes. A acurácia aqui será calculada como o número de acertos dividido pelo número total de casos (24 por dia). Os dados utilizados para checar a acurácia foi um novo conjunto de dados de teste entre o período de '2015-10-02' a '2015-12-30', totalizando 94 dias. Abaixo pode-se ver o resultado obtido:



Pode-se notar que existem dias em que o modelo acerta próximo dos 80% (provavelmente dias com tempo estável) e outros em que acerta menos do que 50% dos casos diários, principalmente no final da série, talvez por serem dias de rigoroso inverno em Nova York.

Distribuição Estacionária

O resultado encontrado para a distribuição estacionária foi aproximadamente o seguinte, utilizando-se a fórmula na primeira seção:

$$\pi = [0.20, 0.21, 0.16, 0.14, 0.15]$$

Implementação

Abaixo o código feito em Python:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

df = pd.read_csv('data/weather_description.csv')
df = df[['datetime', 'New York']]
df.columns = ['datetime', 'weather_desc']

df['datetime'] = pd.to_datetime(df['datetime'])
df = df[df['datetime'].between('2015-01-01', '2016-01-06', inclusive=True)]

print('Rows before drop nan:', df.shape[0])
df = df.dropna()
print('Rows after:', df.shape[0])

print(df.describe())
df

#plot descriptions
sns.set()
```

```
sns.catplot(y='weather_desc', kind='count', data=df)
plt.title('New York Hourly Weather Description')
plt.ylabel('Weather Description')
plt.savefig('figs/ny_temp_descriptions.png', dpi=300)

#generalizing weather types
df['weather_desc'].loc[df['weather_desc'].str.contains('clear')]='clean'
df['weather_desc'].loc[df['weather_desc'].str.contains('clouds')]='cloudly'
df['weather_desc'].loc[df['weather_desc'].str.contains(\
                        'rain|drizzle|thunderstorm|squalls')]='rainy'
df['weather_desc'].loc[df['weather_desc'].str.contains('snow')]='snowy'
df['weather_desc'].loc[df['weather_desc'].str.contains('mist|haze|fog|smoke')]='foggy'

#plotting filtered description
sns.catplot(y='weather_desc', kind='count', data=df)
plt.title('New York Hourly Weather Description')
plt.ylabel('Weather Description')
plt.savefig('figs/ny_temp_descriptions_filtered.png', dpi=300)

#function to transform each type into an int number
def transform_desc(df, dim):

    conditions = []
    for weather_desc in df['weather_desc'].unique():
        conditions.append(df['weather_desc']==weather_desc)

    weather_desc_index = np.arange(0, dim)
    df['weather_desc_index'] = np.select(conditions, weather_desc_index)

    return(df)

df = transform_desc(df, 5)


#plotting filtered description
sns.catplot(y='weather_desc_index', kind='count', data=df)
plt.title('New York Hourly Weather Index')
plt.ylabel('Weather Index')
plt.savefig('figs/ny_temp_descriptions_filtered_index.png', dpi=300)

#calculate transition matrix
def transition_matrix(x, dim):

    P = np.zeros((dim, dim))
    N = len(x)

    for i in range(N-1):

        j = x[i]    #current
        k = x[i+1]  #next

        #probability of state k given state j
        P[k][j] += 1 / N

    return(P)

#create train and test dataframes
df['datetime'] = pd.to_datetime(df['datetime'])
```

```
df_train = df[df['datetime'].between('2015-01-01', '2015-10-01')]
df_test = df[df['datetime'].between('2015-10-02', '2016-10-06')]
df_test1 = df[df['datetime'].between('2015-10-25', '2015-10-26')]
df_test2 = df[df['datetime'].between('2016-01-05', '2016-01-06')]

#calculate probs transition matrix
x = list(df_train['weather_desc_index'])
P = transition_matrix(x, 5)
print(P)

#these predictions consider the current state as the actual that ocurred
def make_predictions(x, n_pred, dim):

    predictions = {
        'prediction': [],
        'max_prob': []
    }

    i = 1

    for state in x:

        if(i<=n_pred):

            x_current = np.zeros(dim)
            x_current[state] = 1

            prediction = np.dot(x_current, P)
            prediction = list(prediction)
            max_prob = prediction.index(max(prediction))

            predictions['prediction'].append(prediction)
            predictions['max_prob'].append(max_prob)

        i += 1

    return(predictions)

#making predictions test data 1
from matplotlib.ticker import FuncFormatter, MaxNLocator

x = list(df_test1['weather_desc_index'])

predictions = make_predictions(x, df_test1.shape[0]-1, 5)

fig, ax = plt.subplots(figsize=(12,6))
ax.plot(np.arange(1,len(x)), x[1:], '-o', label='real')
ax.plot(np.arange(1,len(x)),predictions['max_prob'], '-^', color='red', \
            alpha=0.4, label='prediction')
ax.legend()
ax.yaxis.set_major_locator(MaxNLocator(integer=True))
ax.set_xlim([0.5,25])
ax.set_title('Results - Test Data 1')
ax.set_xlabel('Hours Since 2015-10-25 - 00:00:00')
ax.set_ylabel('Weather Type Index')

plt.savefig('figs/result_weather_dftest1.png', dpi=300)
```

```
#making predictions on test data 2
x = list(df_test2['weather_desc_index'])
predictions = make_predictions(x, len(x) -1, 5)

fig, ax = plt.subplots(figsize=(12,6))
ax.plot(np.arange(1,len(x)), x[1:], '-o', label='real')
ax.plot(np.arange(1,len(x)), predictions['max_prob'], '-^', color='red', \
            alpha=0.4, label='prediction')
ax.legend()
ax.yaxis.set_major_locator(MaxNLocator(integer=True))
ax.set_xlim([0.5,25])
ax.set_title('Results - Test Data 2')
ax.set_xlabel('Hours Since 2016-01-05 - 00:00:00')
ax.set_ylabel('Weather Type Index')
plt.savefig('figs/result_weather_dftest2.png', dpi=300)
```

Referências

[1] En.wikipedia.org. 2020. Markov Chain. [online] Available at: https://en.wikipedia.org/wiki/Markov_chain#Stationary_distribution [Accessed 15 June 2020].

[2] Stat.auckland.ac.nz. 2020. [online] Available at: https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf [Accessed 15 June 2020].

[3] Web.math.ku.dk. 2020. [online] Available at: http://web.math.ku.dk/noter/filer/stoknoter.pdf [Accessed 14 June 2020].

[4] Kaggle.com. 2020. Historical Hourly Weather Data 2012-2017. [online] Available at: https://www.kaggle.com/selfishgene/historical-hourly-weather-data?select=weather_description.csv [Accessed 13 June 2020].