TripAdvisor_HTML

August 18, 2022

Pontifícia Universidade Católica do Paraná

Disciplina de Técnicas de Machine Learning

avaliações TRIP ADVISOR —> Base de Dados UCI

Nome do arquivo: tripadvisor_reviews.csv
Fonte Original do arquivo: https://archive.ics.uci.edu/ml/datasets/Travel+Reviews

Esta base de dados (dataset) contém as médias das notas, que diversos usuários do portal Trip Advisor deram a diferentes locais na Ásia.

IMPORTANTE!! O dataset usado foi ligeiramente alterado em relação à versão disponível na fonte original.

Este projeto busca prever, a partir das notas dadas por um usuário em outras categorias, qual seria a nota média que ele daria para as baladas (coluna "media_baladas").

- Entradas: através da biblioteca Pandas está lendo o arquivo csv tripadvisor_reviews.csv
- Saídas: uma predição utilizando o modelo nao supervisionado KNN (apresentou a maior precisão nos testes).
- Período: 980 usuários X 10 categorias, coletados em diferentes locais da Ásia.
- Objetivo: buscando prever a nota media para um item, segundo as notas dadas por esse usuario para outros itens.
- Autoria: desenvolvido por Victor Marques - PUCPR - 30.10.2021

Preparando o sistema, importando as Bibliotecas...

[130]:
```
# importando as bibliotecas usadas
import pandas as pd
import numpy as np
```

e a base de dados.

[131]:
```
# importando a base de dados TripAdvisor
df = pd.read_csv('tripadvisor_reviews.csv' , sep=';')
```

Assim ela está:

[132]:
```
df[10:39:3]
```

[132]:

	usuario	media_galerias_arte	media_baladas	media_loja_sucos	\
10	User 11	1,47	1	0,7	
13	User 14	0,58	1,64	2,27	
16	User 17	0,86	1,04	1,76	
19	User 20	0,8	1,04	2,1	
22	User 23	0,93	1,16	0,29	
25	User 26	0,61	2,84	2,8	
28	User 29	0,58	1,2	0,18	
31	User 32	0,7	2,24	2,32	
34	User 35	0,96	1,16	0,45	
37	User 38	1,02	1,36	0,91	

	media_restaurantes	media_museus	media_resorts	media_parques	media_praias	\
10	0,75	1,66	2,76	3,18	2,89	
13	0,45	1,26	1,72	3,19	2,91	
16	0,34	0,06	1,1	3,18	2,73	
19	0,58	1,18	1,98	3,19	2,93	
22	0,41	1,02	1,36	3,16	2,74	
25	0,48	0,56	1,52	3,19	2,54	
28	0,38	0,54	0,76	3,17	2,69	
31	0,63	0,72	2,12	3,19	2,65	
34	0,29	0,98	1,42	3,18	2,94	
37	0,5	0,72	1,22	3,18	2,91	

	media_teatros	media_templos_religiosos
10	1,66	2,62
13	2,3	2,74
16	1,15	2,98
19	1,22	2,48
22	1,34	3,66
25	1,6	2,54
28	1,63	2,94
31	1,28	2,42
34	2,02	3,02
37	1,92	3,2

Então mudamos os nomes da colunas e removemos a coluna usuário, que não importa para o Aprendizado da Máquina...

[133]:
```
df.drop(columns=['usuario'], inplace=True)
df.columns = df.columns.str.replace('media_', '').str.title().str.replace('_',␣
↪' ')
display(df.columns)
```

```
Index(['Galerias Arte', 'Baladas', 'Loja Sucos', 'Restaurantes', 'Museus',
       'Resorts', 'Parques', 'Praias', 'Teatros', 'Templos Religiosos'],
      dtype='object')
```

Aqui vemos se a base de dados possui algum campos vazio.

[134]:
```
df.isnull().sum()
```

[134]:
```
Galerias Arte         0
Baladas               0
Loja Sucos            0
Restaurantes          0
Museus                0
Resorts               0
Parques               0
Praias                0
Teatros               0
Templos Religiosos    0
dtype: int64
```

e convertemos vírgulas em pontos para que a função profiling da bibliotecas pandas funcione.

[135]:
```
# convertendo a virgula em ponto
df = df.apply(lambda col: col.str.replace(',', '.').astype(float))
```

[136]:
```
df[10:39:3]
```

[136]:

	Galerias Arte	Baladas	Loja Sucos	Restaurantes	Museus	Resorts
10	1.47	1.00	0.70	0.75	1.66	2.76
13	0.58	1.64	2.27	0.45	1.26	1.72
16	0.86	1.04	1.76	0.34	0.06	1.10
19	0.80	1.04	2.10	0.58	1.18	1.98
22	0.93	1.16	0.29	0.41	1.02	1.36
25	0.61	2.84	2.80	0.48	0.56	1.52
28	0.58	1.20	0.18	0.38	0.54	0.76
31	0.70	2.24	2.32	0.63	0.72	2.12
34	0.96	1.16	0.45	0.29	0.98	1.42
37	1.02	1.36	0.91	0.50	0.72	1.22

	Parques	Praias	Teatros	Templos Religiosos
10	3.18	2.89	1.66	2.62
13	3.19	2.91	2.30	2.74
16	3.18	2.73	1.15	2.98
19	3.19	2.93	1.22	2.48
22	3.16	2.74	1.34	3.66
25	3.19	2.54	1.60	2.54
28	3.17	2.69	1.63	2.94
31	3.19	2.65	1.28	2.42
34	3.18	2.94	2.02	3.02
37	3.18	2.91	1.92	3.20

Analisamos médias, nota mínima e máxima, padrão de desvio, quartis e se todas as categorias tem notas.

[137]:
```
df.describe().round(3)
```

[137]:

	Galerias Arte	Baladas	Loja Sucos	Restaurantes	Museus	Resorts
count	980.000	980.000	980.000	980.000	980.000	980.000
mean	0.893	1.353	1.013	0.532	0.940	1.843
std	0.327	0.478	0.789	0.280	0.437	0.540
min	0.340	0.000	0.130	0.150	0.060	0.140
25%	0.670	1.080	0.270	0.410	0.640	1.460
50%	0.830	1.280	0.820	0.500	0.900	1.800
75%	1.020	1.560	1.572	0.580	1.200	2.200
max	3.220	3.640	3.620	3.440	3.300	3.760

	Parques	Praias	Teatros	Templos Religiosos
count	980.000	980.000	980.000	980.000
mean	3.181	2.835	1.569	2.799
std	0.008	0.138	0.365	0.321
min	3.160	2.420	0.740	2.140
25%	3.180	2.740	1.310	2.540
50%	3.180	2.820	1.540	2.780
75%	3.180	2.910	1.760	3.040
max	3.210	3.390	3.170	3.660

[138]:
```
# !pip install pandas_profiling
```

E levantando maiores detalhes sobre cada categoria.

[139]:
```
from pandas_profiling import ProfileReport # importando o pandas-profiling para␣
↪fazer o profile do dataset
profile = ProfileReport(df)
profile.to_widgets()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render widgets:   0%|          | 0/1 [00:00<?, ?it/s]

VBox(children=(Tab(children=(Tab(children=(GridBox(children=(VBox(children=(GridspecLayout(chil
```

Apesar da análise acima mostrar 37 usuários que apresentaram as mesmas notas e formarem linhas duplicadas, pode-se tomar como base que a coleta das notas apresentava apenas 6 opções de nota, como no sistema de 5 estrelas de classificação normalmente utilizado. Assim seria possível termos tantas notas iguais.

[140]:
```
import seaborn as sns # importando o Seaborn para visualizar o comportamento␣
↪dos dados
# import math
# math.isclose(np.var(df.values), 0)
# sns.pairplot(df, hue="Baladas")
df['Baladas'].tail()
```

```
# df = df.apply(lambda col: col.str.replace(',', '.').astype(int))
# sns.kdeplot(df)
```

[140]:
```
975    1.12
976    0.92
977    1.32
978    0.20
979    0.56
Name: Baladas, dtype: float64
```

[141]:
```
# sns.pairplot(df, hue='Baladas')
```

[142]:
```
df = df.apply(lambda col: col.replace('.', ',').astype(float))
df
```

[142]:

	Galerias Arte	Baladas	Loja Sucos	Restaurantes	Museus	Resorts	Parques	Praias	Teatros	Templos Religiosos
0	0.93	1.80	2.29	0.62	0.80	2.42	3.19	2.79	1.82	2.42
1	1.02	2.20	2.66	0.64	1.42	3.18	3.21	2.63	1.86	2.32
2	1.22	0.80	0.54	0.53	0.24	1.54	3.18	2.80	1.31	2.50
3	0.45	1.80	0.29	0.57	0.46	1.52	3.18	2.96	1.57	2.86
4	0.51	1.20	1.18	0.57	1.54	2.02	3.18	2.78	1.18	2.54
..	…	…	…	…	…	…	…	…	…	…
975	0.74	1.12	0.30	0.53	0.88	1.38	3.17	2.78	0.99	3.20
976	1.25	0.92	1.12	0.38	0.78	1.68	3.18	2.79	1.34	2.80
977	0.61	1.32	0.67	0.43	1.30	1.78	3.17	2.81	1.34	3.02
978	0.93	0.20	0.13	0.43	0.30	0.40	3.18	2.98	1.12	2.46
979	0.93	0.56	1.13	0.51	1.34	2.36	3.18	2.87	1.34	2.40

[980 rows x 10 columns]

[143]:
```
# sns.boxplot(df['Restaurantes'])
# plt.show()
```

[144]:
```
# sns.histplot(df['Baladas'])
```

[145]:
```
# sns.scatterplot(data=df, x="Praias", y="Baladas")
```

Separando entre treinamento e teste (75% treino e 25% de teste)

[146]:
```
from sklearn.model_selection import train_test_split

# split entre treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(df.drop('Baladas', axis=1),
                                                    df['Baladas'],
                                                    test_size=0.25,
                                                    random_state=0)
```

0.1 Treinamento dos modelos

0.1.1 KNN para regressão

[147]:
```
from sklearn.neighbors import KNeighborsRegressor # KNN para regressão
modelo_knn = KNeighborsRegressor().fit(X_train, y_train)
modelo_knn.score(X_test, y_test)
```

[147]: 0.13639483791844376

Regressão linear

[148]:
```
from sklearn.linear_model import LinearRegression # Regressão linear
modelo_lr = LinearRegression().fit(X_train, y_train)
modelo_lr.score(X_test, y_test)
```

[148]: 0.17527295952698618

0.1.2 SVM para regressão

[149]:
```
from sklearn.svm import SVR # SVM para regressão
modelo_svm = SVR().fit(X_train, y_train)
modelo_svm.score(X_test, y_test)
```

[149]: 0.1397236076281443

1 Usando o scaler para padronizar as notas

[150]:
```
from sklearn.decomposition import PCA # PCA como aprendizagem não-supervisionada
from sklearn.preprocessing import RobustScaler # utilizado para que todas as␣
 ↪entradas estejam na mesma escala numérica
# split entre treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(RobustScaler().
 ↪fit_transform(df.drop('Baladas', axis=1)), # aqui informamos os atributos

 ↪df['Baladas'], # aqui informamos as labels e na mesma ordem dos atributos
```

```
↪test_size=0.25, # informamos a porcentagem de divisão da base. Geralmente é␣
↪algo entre 20% (0.20) a 35% (0.35)

↪random_state=0) # aqui informamos um "seed". É um valor aleatório e usado␣
↪para que alguns algoritmos iniciem de forma aleatória a sua divisão.
```

1.1 Treinamento dos modelos

1.1.1 modelo KNN

[151]:
```
modelo_knn = KNeighborsRegressor().fit(X_train, y_train)
modelo_knn.score(X_test, y_test)
```

[151]:
```
0.15473340580316086
```

1.1.2 modelo Regressao Linear

[152]:
```
modelo_lr = LinearRegression().fit(X_train, y_train)
modelo_lr.score(X_test, y_test)
```

[152]:
```
0.1752729595269873
```

1.1.3 modelo SVM

[153]:
```
modelo_svm = SVR().fit(X_train, y_train)
modelo_svm.score(X_test, y_test)
```

[153]:
```
0.20528632913158507
```

1.2 Gerando as predições com o modelo SVM

[154]:
```
# gerando as predições
modelo_svm.predict(X_test).round(3)
```

[154]:
```
array([1.376, 1.284, 1.203, 0.481, 1.453, 1.44 , 1.32 , 1.396, 1.625,
       1.157, 1.491, 1.146, 1.106, 1.236, 1.3  , 1.401, 1.142, 1.335,
       1.118, 1.343, 1.131, 1.191, 1.257, 1.786, 1.319, 1.509, 1.278,
       1.111, 1.093, 1.422, 1.424, 1.054, 1.44 , 1.342, 1.132, 1.357,
       1.536, 1.489, 1.273, 1.253, 0.999, 1.373, 1.341, 1.026, 1.432,
       1.116, 1.849, 1.588, 1.239, 1.325, 1.26 , 1.093, 1.293, 1.735,
       1.257, 1.388, 0.844, 1.296, 1.282, 1.18 , 1.571, 1.272, 1.25 ,
       1.643, 1.429, 1.403, 1.462, 1.319, 1.2  , 1.419, 1.442, 1.355,
       1.106, 1.28 , 1.18 , 1.152, 1.089, 1.198, 1.13 , 0.721, 1.445,
       1.327, 1.356, 1.318, 0.994, 1.152, 1.341, 1.45 , 1.293, 1.403,
       1.164, 1.326, 1.71 , 1.113, 0.398, 1.556, 1.186, 1.289, 1.639,
       1.258, 1.294, 1.268, 1.281, 1.061, 1.344, 1.219, 1.341, 1.349,
       1.287, 1.496, 1.293, 1.063, 1.365, 1.582, 1.089, 1.29 , 1.191,
```

```
       1.252, 1.247, 1.163, 1.31 , 1.417, 1.469, 1.134, 1.159, 1.356,
       1.192, 1.324, 1.575, 1.362, 1.313, 1.128, 1.378, 1.428, 1.519,
       1.524, 1.496, 1.014, 1.169, 1.317, 1.235, 1.217, 1.418, 1.057,
       1.457, 1.282, 1.252, 1.702, 1.571, 1.384, 1.356, 1.116, 1.348,
       1.313, 1.672, 1.374, 1.026, 0.898, 1.096, 1.429, 1.371, 1.273,
       0.993, 1.378, 0.664, 1.009, 1.048, 1.113, 1.404, 1.685, 1.239,
       1.21 , 1.267, 1.192, 2.221, 1.241, 1.441, 1.203, 1.163, 1.375,
       1.422, 1.23 , 1.149, 1.21 , 1.376, 1.463, 1.291, 1.227, 1.212,
       1.253, 1.329, 1.244, 1.817, 1.17 , 1.347, 1.521, 1.292, 1.187,
       1.428, 0.743, 1.34 , 1.054, 1.299, 1.193, 1.435, 1.375, 1.344,
       1.553, 1.122, 0.976, 1.107, 1.356, 1.39 , 1.345, 1.404, 1.763,
       1.466, 1.173, 1.457, 1.355, 0.962, 1.547, 1.381, 1.191, 1.446,
       1.193, 1.151, 0.936, 1.408, 1.465, 1.618, 1.267, 1.378, 1.237,
       1.3  , 1.589, 1.265, 1.777, 0.974, 1.138, 1.248, 1.114, 1.048,
       1.162, 1.218])
```

Exibindo um quadro com os resultados reais, de teste e os previstos em 3 modelos de regressão.

[155]:
```
df_test = pd.DataFrame(X_test)
df_test['Real'] = df['Baladas'][705:950].values
df_test['Teste'] = y_test.values
df_test['Predição SVM'] = modelo_svm.predict(X_test)
df_test['KNN'] = modelo_knn.predict(X_test)
df_test['Linear'] = modelo_lr.predict(X_test)
df_test.round(2)
```

[155]:

	0	1	2	3	4	5	6	7	8	Real	Teste	\
0	0.29	0.08	-0.24	-0.86	-0.41	0.00	-0.71	0.71	0.40	1.52	1.28	
1	-0.26	0.72	0.24	0.14	-0.03	0.01	-0.94	-0.71	-0.88	0.96	2.72	
2	-0.46	0.83	-0.29	1.00	0.08	0.00	-0.76	-1.22	-0.08	0.76	1.36	
3	2.37	-0.30	0.53	-0.18	-1.00	0.00	0.76	-0.36	-0.60	0.20	0.24	
4	-0.46	-0.43	0.35	-0.50	-0.73	0.00	-0.06	0.00	0.24	1.44	1.48	
..	…	…	…	…	…	…	…	…	…	…	…	
240	-0.46	0.06	0.00	0.43	0.46	0.00	2.24	-0.51	0.08	0.88	1.20	
241	-0.26	1.48	-0.29	0.14	-0.08	0.02	0.29	-0.36	-0.88	1.48	1.44	
242	0.00	0.15	-0.65	-0.36	-0.65	0.00	0.29	-0.51	-0.64	1.28	1.36	
243	-0.91	0.91	-0.59	-0.75	0.38	0.01	-1.53	-0.87	-0.76	2.12	1.28	
244	-0.17	-0.51	0.53	0.07	0.19	0.00	1.41	-0.29	-0.24	1.72	1.40	

	Predição SVM	KNN	Linear
0	1.38	1.62	1.42
1	1.28	1.23	1.41
2	1.20	1.46	1.33
3	0.48	0.71	0.95
4	1.45	1.38	1.45
..	…	…	…
240	1.25	1.29	1.23

```
241          1.11  1.19    1.33
242          1.05  1.58    1.14
243          1.16  1.46    1.53
244          1.22  1.29    1.31

[245 rows x 14 columns]
```

1.3 Conclusões

```
Para este projeto do Banco de Dados do Trip Advisor, foi utilizada a aprendizagem supervisionad

O problema proposto foi resolvido usando técnicas incluídas no Scikit-learn, Analogizers de reg

Observou-se que ao utilizar o escaler para padronizar as notas médias, o método SVM apresentou
```

obs.: Encontrou-se nesta base de dados uma dificuldade em representar graficamente os dados e os resultados, em virtude de ora bibliotecas faltando ora incompatibilidades que necessitam um aprimoramento maior por parte da equipe envolvida. Tais partes, bibliotecas do seaborn, estão comentadas no código acima para o não impedimento do funcionamento da ML, e ficarão para uma correção futura.