Instituto Tecnológico de Aeronáutica – ITA

Graduação

CTC17 : Inteligência Artificial - Projeto 2

Bruno De Souza Neves , Rahyan Azin

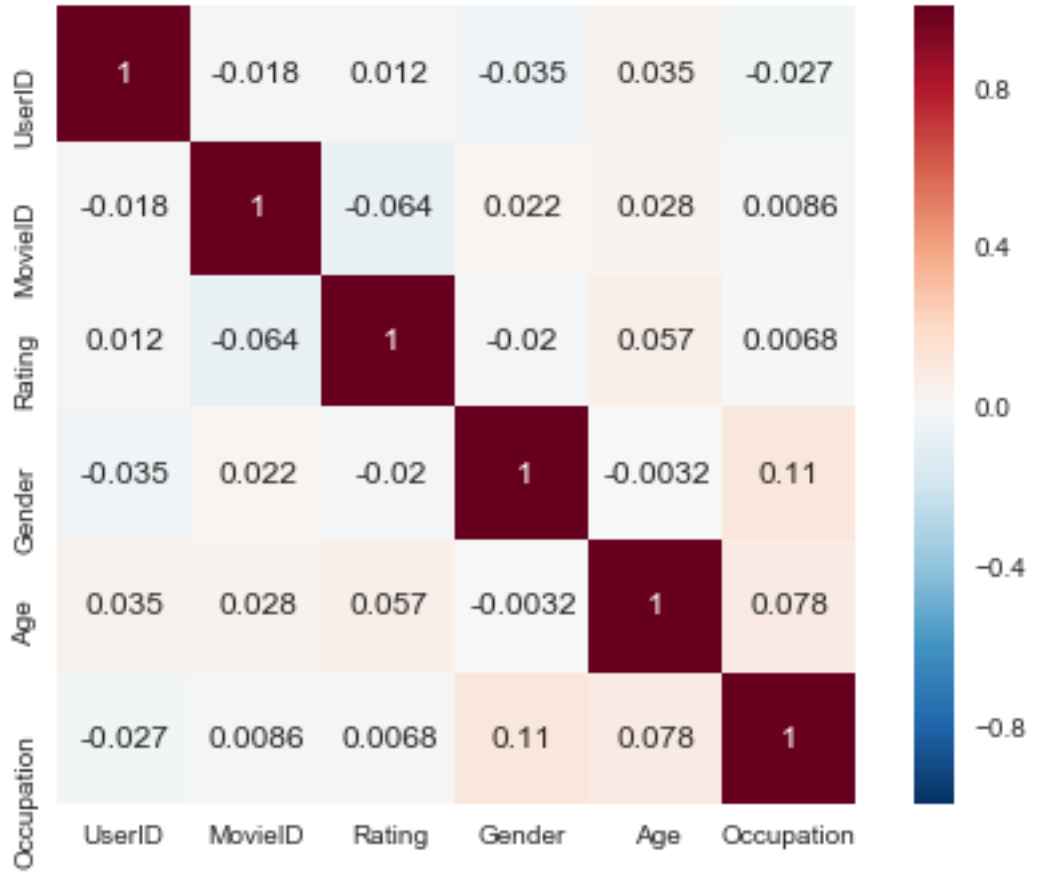
27 de setembro de 2018

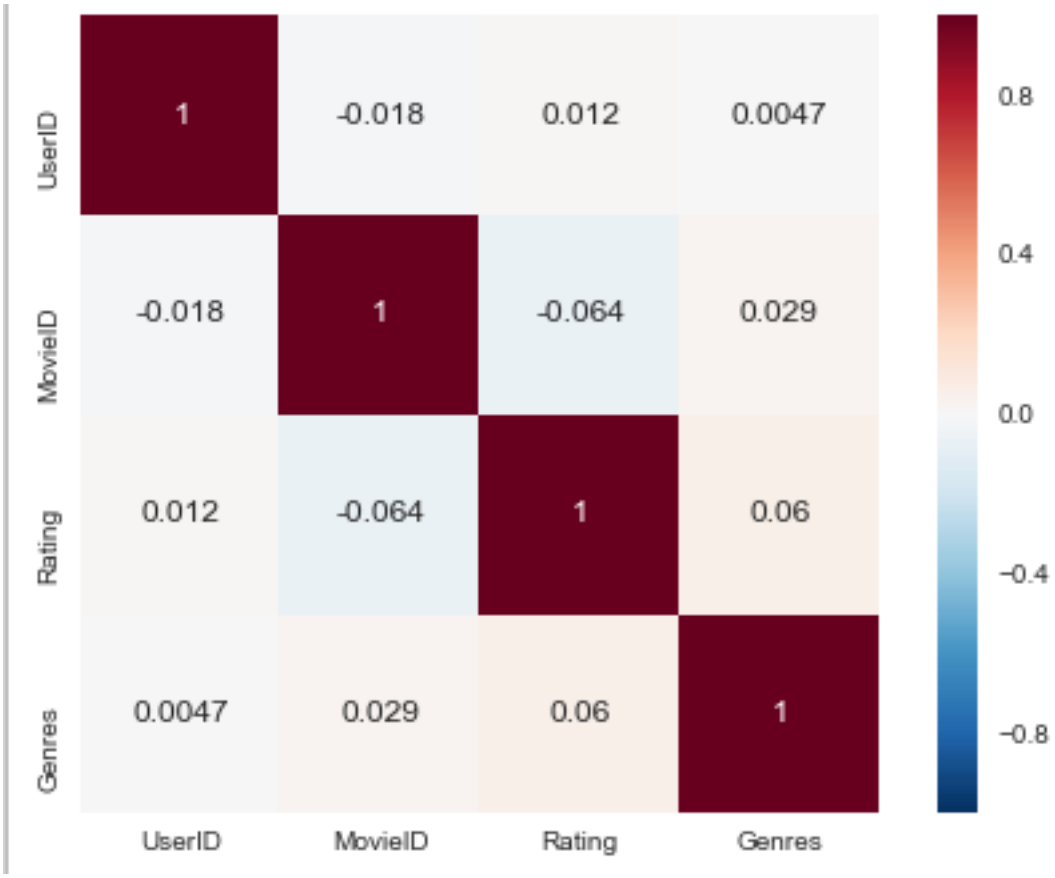
1 Naive Bayes

O algoritmo escolhido para treinamento do agente foi o Naive Bayes , já que a implementação da árvore de decisão se mostrou mais complexa . A linguagem escolhida foi Python por ter bibliotecas, como pandas e numpy , que facilitam a manipulação dos dados . Para a escolha de quais tipos de dados eram mais relevantes no treinamento , foi feito um heatmap , ou matrix de correlação , para analisar quais features eram mais correlacionada com a saída Rating.

```
movie_data = pd.read_csv("movies.dat", sep="::", header=None,
    names=['MovieID','Title','Genres'],
                          dtype={'MovieID': np.int32, 'Title': np.str, 'Genres':
                              np.str}, engine='python')


users_data = pd.read_csv("users.dat", sep="::", header=None,
    names=['UserID','Gender','Age','Occupation','Zip-code'],

                          dtype={'UserID': np.int32, 'Gender': np.str, 'Age': np.int32,
                              'Occupation' : np.int32, 'Zip-code' : np.str},
                              engine='python')

ratings_data = pd.read_csv("ratings.dat",
                          sep="::", header=None,
                              names=['UserID','MovieID','Rating','Timestamp'],
                          dtype={'UserID': np.int32, 'MovieID': np.int32, 'Rating':
                              np.int32, 'Timestamp' : np.str}, engine='python')

data = pd.merge(ratings_data, users_data, how='left', left_on=['UserID'],
    right_on=['UserID'])
data2 = pd.merge(ratings_data, movie_data, how='left', left_on=['MovieID'],
    right_on=['MovieID'])

data = data.head(60000)
data2 = data2.head(60000)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()


le.fit(data['Gender'])
x_g = le.transform(data['Gender'])
data['Gender'] = x_g

le.fit(data2['Genres'])
x_genrer = le.transform(data2['Genres'])
data2['Genres'] = x_genrer
corr = data.corr()
corr2 = data2.corr()
sns.heatmap(corr,annot = True, center=0, square=True)
sns.heatmap(corr2,annot = True,center=0,square=True)
```

Leu-se os dados,juntou-se as tabelas com os comandos merge , codificamos as strings Gender e Genrer e obtivemos os heatmaps abaixo :





Foram escolhidas as features MovieID,Gender,Age,Genrer para o cálculo das probabilidades , baseados em suas correlações com a saída.

	Age	Gender	MovieID	Genres
0	1	0	1193	239
1	1	0	661	152
2	1	0	914	282
3	1	0	3408	239
4	1	0	2355	145
5	1	0	1197	13
6	1	0	1287	19
7	1	0	2804	185
8	1	0	594	152
9	1	0	919	113
10	1	0	595	152
11	1	0	938	281
12	1	0	2398	239
13	1	0	2918	176
14	1	0	1035	281
15	1	0	2791	176
16	1	0	2687	144
17	1	0	2018	144
18	1	0	3105	239
19	1	0	2797	192
20	1	0	2321	176
21	1	0	720	143
22	1	0	1270	211
23	1	0	527	262
24	1	0	2340	290
25	1	0	48	153
26	1	0	1097	169
27	1	0	1721	252
28	1	0	1545	239
29	1	0	745	155

Em seguida , implementou-se o Naive Bayes em duas funções. Uma que calculas as probabilidades condicionais , baseadas nas features , e outra que retorna a Rating prevista de um novo dado.

```
def calculatePriors(training, outcome):
    features = np.unique(outcome)
    rows, cols = np.shape(training)
    similarities = {}
    for ftres in features:
        similarities[ftres] = defaultdict(list)

    class_probabilities = probabilities(outcome)

    for ftres in features:
        row_indices = np.where(outcome == ftres)[0]
        subConjunto = training[row_indices, :]
        r, c = np.shape(subConjunto)
        for j in range(0,c):
            similarities[ftres][j] += list(subConjunto[:,j])

    for ftres in features:
        for j in range(0,cols):
             similarities[ftres][j] = probabilities(similarities[ftres][j])

    return class_probabilities,similarities,features



def predictRating(class_probabilities,similarities,features,newGuy):
    results = {}
    for ftres in features:
```

```
        probs = class_probabilities[ftres]
        for i in range(len(newGuy)):
            relative_values = similarities[ftres][i]
            if newGuy[i] in relative_values.keys():
                probs *= relative_values[newGuy[i]]
            else:
                probs *= 0
            results[ftres] = probs
    return max(results.iteritems(), key=operator.itemgetter(1))[0]
```

Depois dividiu-se o dataset em duas partes:treinamento e test:

```
from sklearn.model_selection import train_test_split
features = data[['Age','Gender','MovieID']]
features['Genres'] = data2['Genres']
outcome = data['Rating']
x_train, x_test, y_train, y_test = train_test_split(features, outcome, test_size=0.25)

class_probabilities,similarities,features =
    calculatePriors(x_train.values,y_train.values)
nTotal = len(x_test)
y_T = y_test.values
x_T = x_test.values
predicteds = [predictRating(class_probabilities,
                            similarities,features,
                            x_T[i]) for i in range(nTotal)]
predicteds = np.asarray(predicteds)

print accuracy_score(y_T, predicteds)
print confusion_matrix(y_T,predicteds)
print cohen_kappa_score(y_T, predicteds)
print mean_squared_error(y_T,predicteds)
```

Finalmente, verificamos as métricas de desempenho :

```
print accuracy_score(y_test, predicteds)
print confusion_matrix(y_T,predicteds)
print cohen_kappa_score(y_T, predicteds)
print mean_squared_error(y_T,predicteds)
```

Que nos deu como resultado:

```
0.3938405058127677
[[ 3512  1653  4711  3538   939]
 [ 2946  2394 10289 9255  2084]
 [ 2892  3073 20944 30068 7947]
 [ 1652  1707 17519 47456 18768]
 [  545   453  5334 26199 24175]]
0.15938378073726223
```

O que nos leva a crer que o algoritmo é razoável quanto à predição , já que o coeficiente Kappa não foi muito alto , mas a taxa de acerto e a quantidade de positivos e negativos verdadeiros foram boas.

Para o classificador a priori :

```
#classificador a priori
    dic = {}
```

```
l = int(0.75*len(data['MovieID']))

for i in range(l):
    p = int(data['MovieID'][i])
    if p not in dic:
      dic[p] =(int(data['Rating'][i]),1)
    else:
        p1 = int(dic[p][0]) + int(data['Rating'][i])
        p2 = int(dic[p][1]) + 1
        dic[p] = (p1,p2)


prioriResult = {}
for i in range(l):
    p = int(data['MovieID'][i])
    if dic[p][1]!=0:
     prioriResult[p] = dic[p][0]/dic[p][1] #classificacao eh a media truncada

results = []
for i in range(l,len(data['MovieID'])):
    p = int(data['MovieID'][i])
    if p in prioriResult:
     results.append(prioriResult[p])
    else:
        results.append(-1)

y_test2 = data['Rating'][l:]
y_test2 = y_test2.values
results = np.asarray(results)

print results,y_test2

print accuracy_score(y_test2, results)
print confusion_matrix(y_test2,results)
print cohen_kappa_score(y_test2, results)
print mean_squared_error(y_test2,results)
```

Resultado:

```
0.3076347814263376
[[  916  5303  6252   936     0]
 [  476  7020 15796 2169     0]
 [  306  9235 44718 9725     1]
 [  106  5302 58920 24271    3]
 [   37  1312 28343 28906    0]]
0.060529535570420734
1.2782650078183424
```

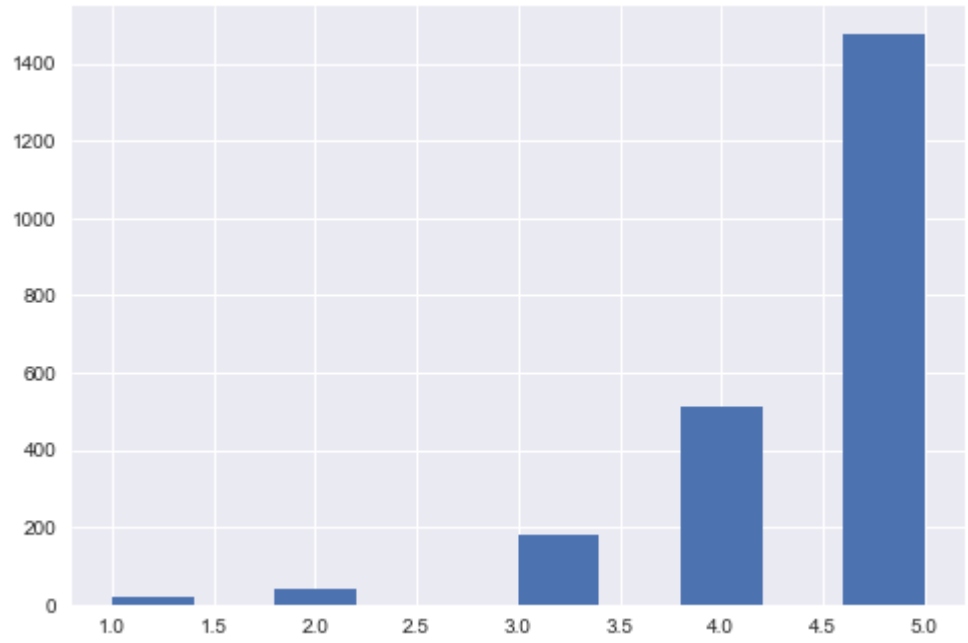
Um resultado menos preciso mas não tão menos preciso que o Naive Bayes . Ou seja, se um filme realmente é bom , então provavelmente uma pessoa que nunca viu vai gostar, dado que uma grande quantidade de pessoas gostou. As métricas apontam para um melhor resultado do Naive Bayes.

2 Análise Resultados

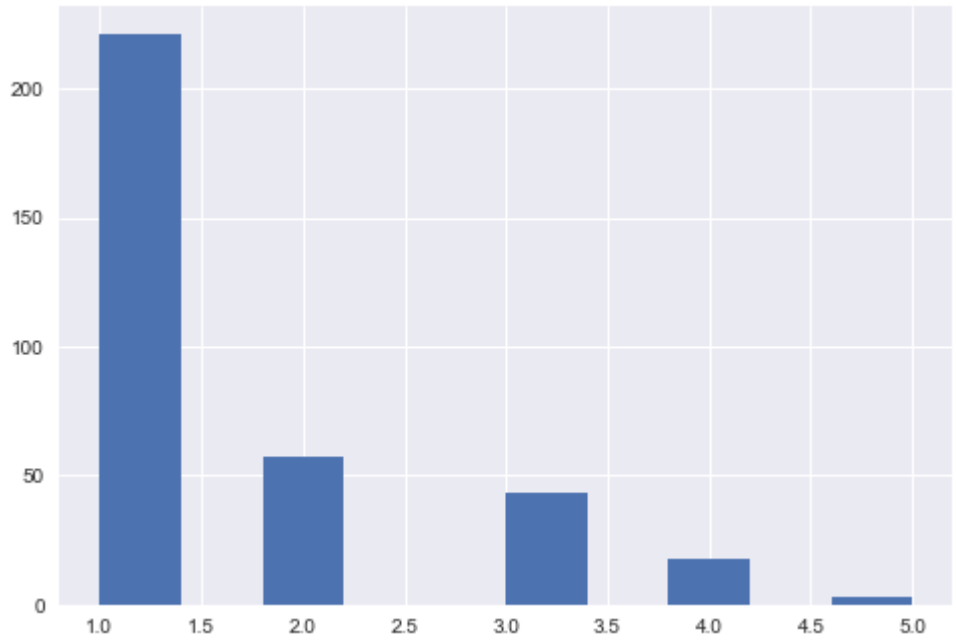
Seria interessante rever a hipótese da independencia de ratings dos usuários , já que , por exemplo , o fato de amigos próximos gostarem de um filme e me persuadirem que ele bom , provavelmente minha classificação para a Rating muda se não ouvisse opinião de outras pessoas. Entretanto,o Naive Bayes se mostrou adequado. O classificador a priori não apresentou um resultado ruim , o motivo é que , intuitivamente , um filme que é muito bom vai ter muitas Ratings 5 , então provavelmente uma pessoa que nunca o viu vai dar a mesma nota que a média.

Por exemplo , analisemos o filme 'O Poderoso Chefão (The Godfather) '

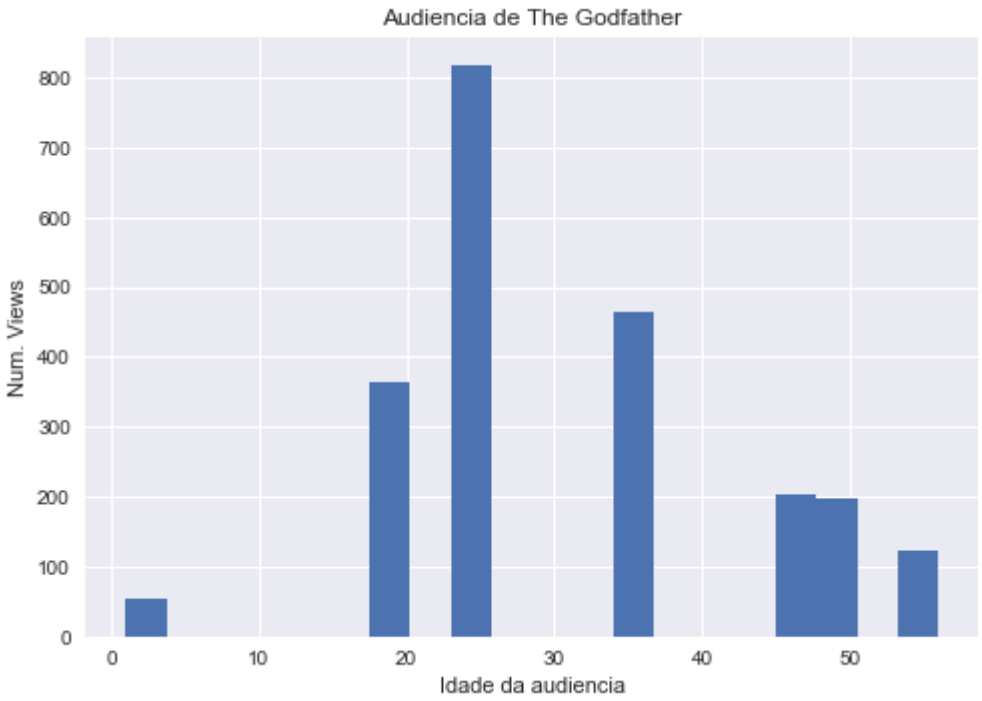
```
godfather_data = ratings_data[ratings_data.MovieID==858]
godfather_data.groupby('Rating').size()
godfather_data_group = godfather_data.groupby('Rating')
godfather_data_group.agg({'Rating':'mean'})
plt.hist(x=godfather_data['Rating'])
plt.show()
```



Ou seja, alguem que viu o poderoso chefão dificilmente vai dar uma nota que não seja 4 ou 5. O mesmo vale para filmes muito ruins, como "Battlefield Earth ":




```
viewership = pd.merge(ratings_data, users_data, how='left', left_on=['UserID'],
    right_on=['UserID'])
viewership_of_toystory = viewership[viewership['MovieID'] == 858]
plt.hist(x=viewership_of_toystory['Age'], data=viewership_of_toystory, bins=20)
plt.xlabel("Idade da audiencia")
plt.ylabel("Num. Views")
plt.title("Audiencia de The Godfather")
plt.show()
```



E também o publico que mais assiste está concentrado entre 20 e 30 anos , o que torna a média uma boa predição , assim como o Naive Bayes.

A mesma análise foi feita com filmes como Toy Story,Star Wars Episodio IV , o que deu resultados semelhantes aos de cima, já que também são filmes bons.

3 Conclusão

O trabalho foi bastante adequado para compreensão do Naive Bayes e também sobre comparação de métodos de aprendizagem , já que a complexidade de programação de um algoritmo em relação a um mais simples , como a classificação por média , tem que fornecer resultados melhores. Trabalhos adicionais poderiam ser: Continuação da análise exploratória de dados, analisar de há overfitting ou underfitting , analisar outros métodos de aprendizagem, etc.