Trabalho de Mineração de Dados

ECO904 - Prof. Carlos Henrique Valério Moraes

Nome: Caíque Cléber Dias de Rezende

Matrícula: 2016003750

1. Introdução

Este trabalho visa ser um relatório explicando o código fonte "*trabalho_04.py*". O código fonte, feito em linguagem Python, recebe a base de dados do arquivo "*aluguel_brasil.csv*", realizando uma extração de agrupamentos. Após coletados, os dados são tratados pelo algoritmo e por fim testa-se a eficácia de diferentes métodos de clusterização na mineração de dados.

Ambos os arquivos citados foram entregues junto a este documento.

2. Definição da Base de Dados (Dataset)

O arquivo "*aluguel_brasil.csv*" representa nossa base de dados e possui 10.962 linhas com informações de imóveis para alugar em cinco cidade brasileiras: Belo Horizonte, São Paulo, Campinas, Porto Alegre e Rio de Janeiro.

Cada uma das linhas da base de dados contém as seguintes colunas e descrições:

- city - cidade onde o imóvel está localizado
- area - área do imóvel
- rooms - número de quartos
- bathroom - número de banheiros
- parking spaces - número de vagas na garagem
- floor - andar para edifícios, '-' para casas
- animal - permite animais no imóvel
- furniture - residência mobiliada
- hoa (R$) - valor do condomínio
- rent amount (R$) - valor do aluguel
- property tax (R$) - IPTU
- fire insurance (R$) - seguro contra incêndio
- total (R$) - valor total pago

3. Apresentação das técnicas a serem utilizadas

As técnicas de clusterização a serem implementadas no algoritmo são as seguintes:

- K-Means
- Mean Shift
- Espectral
- Ward
- Hierarquico
- DBSCAN
- Birch
- Gaussian Mixture

Visa-se criar uma tabela comparativa entre as oito técnicas, possibilitando-se concluir quais delas são mais ou menos eficazes em sua performance.

4. Algoritmo: preparação dos dados

Primeiramente, o código fonte carrega a base de dados diretamente do arquivo "*aluguel_brasil.csv*" para a variável "aluguel":

```
34  # carrega dataset em aluguel
35  aluguel = pd.read_csv('aluguel_brasil.csv')
```

Após carregado, sorteia-se, entre as cinco cidades disponíveis na base de dados, apenas uma para ser analisada. O sorteio feito no código fonte usa um gerador de semente fixado pelo número de matrícula.

```
41  # fixar semente
42  matricula = 2016003750
43  random.seed(matricula)
44
45  # escolha da cidade
46  cidades = ['Belo Horizonte', 'Campinas',
47             'Porto Alegre', 'Rio de Janeiro',
48             'São Paulo']
49  selCidade = random.choice(cidades)
50  print("\n\t---> CIDADE ESCOLHIDA PARA ALUGUEL:", selCidade.upper())
51  ehCidade = aluguel['city'] == selCidade
52  aluguel = aluguel[ehCidade]
53  aluguel = aluguel.drop('city', axis=1)
54  print("\nTABELA PARA A CIDADE:\n")
55  print(aluguel.head())
```

Na execução do algoritmo, a cidade escolhida para o número de matrícula em questão foi “São Paulo”. A tabela a seguir mostra as primeiras linhas da base de dados filtradas apenas por essa cidade:

```
      ---> CIDADE ESCOLHIDA PARA ALUGUEL: SÃO PAULO

TABELA PARA A CIDADE:

   area  rooms  bathroom  parking spaces floor     animal     furniture  hoa (R$)  rent amount (R$)  property tax (R$)  fire insurance (R$)  total (R$)
0    70      2         1               1     7      acept     furnished      2065              3300                211                   42        5618
1   320      4         4               0    20      acept  not furnished      1200              4960               1750                   63        7973
4    25      1         1               0     1  not acept  not furnished         0               800                 25                   11         836
5   376      3         3               7     -      acept  not furnished         0              8000                834                  121        8955
7   213      4         4               4     4      acept  not furnished      2254              3223               1735                   41        7253
```

A seguir, o algoritmo sorteia três colunas aleatórias, removendo-as da base de dados.

```
55  # remover 3 colunas aleatórias de aluguel
56  colunas = aluguel.columns.values.copy()
57  random.shuffle(colunas)
58  colunas = colunas[:3]
59  print("\n\t---> COLUNAS SORTEADAS:", colunas)
60  aluguel = aluguel.drop(colunas, axis=1)
61  print("\nTABELA PARA A CIDADE SEM AS COLUNAS SORTEADAS:\n")
62  print(aluguel.head())
63
64  # contar número de imóveis em aluguel
65  print("\n\t---> NÚMERO DE IMÓVEIS = ", len(aluguel))
```

As colunas sorteadas bem como uma nova tabela com as primeiras linhas da base de dados e também o número de imóveis disponíveis para aluguel na cidade de São Paulo são mostradas durante a execução:

```
      ---> COLUNAS SORTEADAS: ['property tax (R$)' 'floor' 'area']

TABELA PARA A CIDADE SEM AS COLUNAS SORTEADAS:

   rooms  bathroom  parking spaces     animal     furniture  hoa (R$)  rent amount (R$)  fire insurance (R$)  total (R$)
0      2         1               1      acept     furnished      2065              3300                   42        5618
1      4         4               0      acept  not furnished      1200              4960                   63        7973
4      1         1               0  not acept  not furnished         0               800                   11         836
5      3         3               7      acept  not furnished         0              8000                  121        8955
7      4         4               4      acept  not furnished      2254              3223                   41        7253

      ---> NÚMERO DE IMÓVEIS =  5887
```

Embora quase todas as colunas da base de dados original sejam colunas numéricas, as colunas “floor”, “animal” e “furniture” são originalmente colunas de valores textuais. Para que os métodos de clusterização consigam tratar a base de dados da forma

como será apresentado adiante no código fonte, é necessário que todas suas colunas sejam numéricas.

Sabendo-se que já foram sorteadas três colunas aleatórias e removidas da base de dados, verifica-se se, entre as colunas que foram removidas, estão presentes alguma das colunas textuais.

```
71  # verificar se as colunas 'floor', 'animal' ou 'furniture'
72  # foram sorteadas para exclusão
73  removed_floor = False
74  removed_animal = False
75  removed_furniture = False
76  for i in range(len(colunas)):
77      if(colunas[i] == 'floor'):
78          removed_floor = True
79      if(colunas[i] == 'animal'):
80          removed_animal = True
81      if(colunas[i] == 'furniture'):
82          removed_furniture = True
83
```

Caso alguma das três colunas textuais ainda esteja presente na base de dados, então faz-se necessário convertê-la para uma coluna numérica:

```
84  # transformar as colunas de texto ('floor', 'animal', 'furniture')
85  # em colunas numéricas, caso NÃO tenham sido removidas de aluguel
86  labelEncoder = LabelEncoder()
87  if(removed_floor == False):
88      labelEncoder.fit(aluguel['floor'])
89      aluguel['floor'] = labelEncoder.transform(aluguel['floor'])
90  if(removed_animal == False):
91      labelEncoder.fit(aluguel['animal'])
92      aluguel['animal'] = labelEncoder.transform(aluguel['animal'])
93  if(removed_furniture == False):
94      labelEncoder.fit(aluguel['furniture'])
95      aluguel['furniture'] = labelEncoder.transform(aluguel['furniture'])
96  print("\nTABELA PARA A CIDADE SEM AS COLUNAS SORTEADAS E COM VALORES APENAS NUMÉRICOS:\n")
97  print(aluguel.head())
```

A nova tabela é mostrada a seguir. Observe que, embora a coluna “floor” tenha sido excluída da base de dados nos passos anteriores, ainda restaram as colunas “animal” e

```
TABELA PARA A CIDADE SEM AS COLUNAS SORTEADAS E COM VALORES APENAS NUMÉRICOS:

   rooms  bathroom  parking spaces  animal  furniture  hoa (R$)  rent amount (R$)  fire insurance (R$)  total (R$)
0      2         1               1       0          0      2065              3300                   42        5618
1      4         4               0       0          1      1200              4960                   63        7973
4      1         1               0       1          1         0               800                   11         836
5      3         3               7       0          1         0              8000                  121        8955
7      4         4               4       0          1      2254              3223                   41        7253
```

“furniture”, que agora apresentam números da melhor forma que o algoritmo pôde converter.

Por fim, normaliza-se a base de dados antes que ela possa ser aplicada nas técnicas de estudo do trabalho:

```
 97  # normalizando dataset
 98  X = StandardScaler().fit_transform(aluguel)
 99  print("\nDATASET NORMALIZADO:\n")
100  print(X)
```

5. Algoritmo: técnicas de clusterização

A definição de cada técnica de clusterização a ser utilizada no algoritmo é relativamente simples e mostrada no trecho a seguir.

```
101
102  # técnicas de agrupamento, NECESSÁRIO ESTUDAR E OTIMIZAR OS PARÂMETROS DE CADA TÉCNICA
103  two_means = cluster.MiniBatchKMeans(
104      n_clusters=2, init='random', n_init=10, max_iter=300, tol=1e-04, random_state=0)
105
106  bandwidth = cluster.estimate_bandwidth(X, quantile=0.95)
107  ms = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
108
109  spectral = cluster.SpectralClustering(
110      n_clusters=8, eigen_solver='arpack', affinity="nearest_neighbors", random_state=0)
111
112  connectivity = kneighbors_graph(X, n_neighbors=10, include_self=False)
113  connectivity = 0.5 * (connectivity + connectivity.T)
114  ward = cluster.AgglomerativeClustering(
115      n_clusters=3, linkage='ward', connectivity=connectivity)
116  average_linkage = cluster.AgglomerativeClustering(
117      linkage="average", affinity="cityblock", n_clusters=3, connectivity=connectivity)
118
119  dbscan = cluster.DBSCAN(eps=5)
120  birch = cluster.Birch(n_clusters=3, threshold=0.7)
121  gmm = mixture.GaussianMixture(n_components=2, max_iter=300, random_state=0, n_init=10)
122
123  clustering_algorithms = (
124      ('KMeans', two_means),
125      ('MeanShift', ms),
126      ('Espectral', spectral),
127      ('Ward', ward),
128      ('Hierarquico', average_linkage),
129      ('DBSCAN', dbscan),
130      ('Birch', birch),
131      ('GaussianMixture', gmm)
132  )
133
```

Observe que cada uma das técnicas pode receber diversos parâmetros, sendo que a correta aplicação de cada parâmetro é a real dificuldade da utilização das técnicas e alvo de diversos artigos estudando métodos para obter os parâmetros ideias em cada tipo de problema. Por este motivo, os parâmetros aqui definidos foram obtidos de forma empírica.

Para medir a performance das técnicas, o algoritmo as compara criando uma tabela e usando os critérios de tempo em segundos, grupos, ruídos e coeficiente de silhueta, sendo este um coeficiente que pode variar de -1.00 no pior caso a até 1.00 para o melhor caso.

```
134  # impressão de resultados
135  print("\nTABELA DE RESULTADOS PARA AS TÉCNICAS DE AGRUPAMENTO:\n")
136  table = PrettyTable()
137  table.field_names = ["Algoritmo",
138                       "Tempo(s)", "Grupos", "Ruidos", "Coef.Silhueta"]
139
140  for name, algorithm in clustering_algorithms:
141      t0 = time.time()
142      with warnings.catch_warnings():
143          warnings.filterwarnings("ignore", category=UserWarning)  # oculta erros
144          algorithm.fit(X)
145      t1 = time.time()
146
147      if hasattr(algorithm, 'labels_'):
148          grupos = algorithm.labels_.astype(np.int)
149      else:
150          grupos = algorithm.predict(X)
151
152      numGrupos = len(set(grupos)) - (1 if -1 in grupos else 0)  # ignora ruídos
153      numRuido = list(grupos).count(-1)
154      silhueta = metrics.silhouette_score(X, grupos)
155
156      table.add_row([name, '%.2f' % (t1 - t0), numGrupos,
157                     numRuido, '%.2f' % silhueta])
158
159  print(table)
160
```

A execução do trecho acima retorna a tabela principal do trabalho, nosso objetivo:

```
TABELA DE RESULTADOS PARA AS TÉCNICAS DE AGRUPAMENTO:

+-----------------+----------+--------+--------+--------------+
|    Algoritmo    | Tempo(s) | Grupos | Ruidos | Coef.Silhueta |
+-----------------+----------+--------+--------+--------------+
|      KMeans     |   0.05   |   2    |   0    |     0.37     |
|    MeanShift    |   0.29   |   5    |   0    |     0.73     |
|    Espectral    |   4.09   |   8    |   0    |     0.06     |
|       Ward      |   0.59   |   3    |   0    |     0.31     |
|   Hierarquico   |   0.53   |   3    |   0    |     0.91     |
|      DBSCAN     |   1.56   |   1    |   4    |     0.92     |
|      Birch      |   0.38   |   3    |   0    |     0.91     |
| GaussianMixture |   0.89   |   2    |   0    |     0.30     |
+-----------------+----------+--------+--------+--------------+
```

É da tabela acima que definiu-se cada um dos parâmetros das técnicas utilizadas: a cada execução do algoritmo final, tentou-se fazer com que a tabela apresentasse os menores valores de erros e os coeficientes de silhuetas mais próximos possíveis de 1.00.

6. Conclusões

De acordo com a tabela de resultados para as técnicas de agrupamento, pode classificar as técnicas de clusterização aplicadas no algoritmo de acordo com diferentes critérios:

1. Mais veloz: KMeans
2. Mais lenta: Espectral
3. Maior coeficiente de silhueta: DBSCAN
4. Menor coeficiente de silhueta: Espectral
5. Maior número de grupos: Espectral
6. Menor número de grupos: DBSCAN

Vale ressaltar, entretanto, que os valores obtidos por cada técnica são totalmente dependentes dos parâmetros que recebem na sua declaração dentro do algoritmo. Como os parâmetros não foram ajustados com métodos ideais, então não pode-se afirmar que as classificações acima serão sempre verdadeiras, mas valem como uma base para um primeiro entendimento de como cada uma das técnicas se comportaram no trabalho.