Relatório do Trabalho de Sistemas Inteligentes sobre k-Means e SOMs

Lucas Teixeira Gonçalves
Centro de Ciências Computacionais – C3
Universidade Federal de Rio Grande – FURG
Rio Grande, Brasil
Email: lucas.teixeira@furg.br

Palavras-Chave—Mineração de Dados, Twitter, k-Means

I. Introdução

Neste trabalho foi feita a análise de perfis na plataforma *Twitter*, realizando um agrupamento dos usuários para recomendação de produtos em um site de vendas pela web. Para esta tarefa, foram utilizados dois algoritmos: k-Means e Self-organizing Map (SOM). Ambos os algoritmos possuem o mesmo propósito, que é separar um conjunto de dados em grupos levando em conta suas similaridades.

II. Mineração dos Dados

Os dados utilizados no trabalho foram extraídos de aproximadamente 900 seguidores do perfil *beyonce* na plataforma *Twitter*. O código utilizado para a extração dos seguidores foi:

```
def getUsernames(base_user):
  while total < 900:
    response = session.get(base_url +
              "followers/list.json?
              screen_name=%s&cursor=%s"
              % (base_user, next_cursor))
    full = json.loads(response.content)
    followers += full["users"]
    next_cursor = full["next_cursor_str"]
    total += len(full["users"])
    for user in followers:
      names_list.append(user["screen_name"])
  return names_list
```

onde $base\_user$ é o usuário base para pegar os $followers$ e $names\_list$ é a lista com o nome dos $followers$.

De cada um destes usuários, foram coletados os números de $followers$, $following$ e $tweets$ através da função

```
def getUserInfo(user):
  response = session.get(base_url +
  "users/show.json?screen_name=%s" % user)
  info = json.loads(response.content)
  return info["screen_name"],
         info["followers_count"],
         info["friends_count"],
         info["statuses_count"]
```

onde *user* é o usuário em questão e *info* nos índices *screen_name, followers_count, friends_count, statuses_count* são, respectivamente, o *username*, número de *followers*, número de pessoas que está seguindo e número de *tweets*.

III. k-Means

O algoritmo de k-Means é usado para separar o conjunto de dados em k diferentes grupos. O problema é considerado *NP-HARD*, entretanto, pequenas bases de dados e algoritmos heurísticos fazem com que seja possível uma rápida convergência para um mínimo local, como pode ser visto na Fig 1.

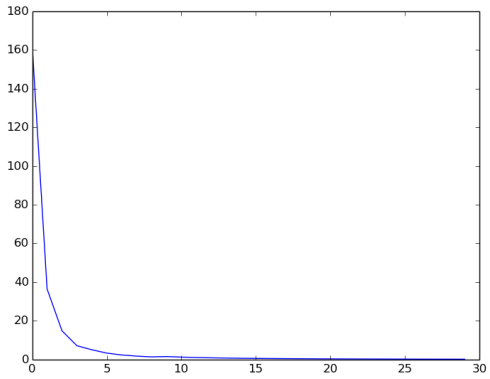

Figura 1. Convergência das médias no algoritmo k-Means

Primeiramente, k valores aleatórios são inicializados e tomados como as médias iniciais dos grupos. Depois, cada amostra é colocada em um grupo, baseado na Distância Euclidiana, representada por

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}. \quad (1)$$

Após o posicionamento de cada amostra nos grupos, uma nova média é calculada para cada grupo, repetindo o processo (sem inicializar as médias novamente) por n épocas.

No trabalho, foram utilizadas $n = 30$ épocas para separar as amostras em $k = 3$ grupos diferentes, resultando numa separação dos grupos que pode ser vista na Fig 2, onde os pontos circulares são as amostras iniciais, as estrelas são as

médias dos grupos e o triângulo é um usuário utilizado para o teste.

Para melhor visualização, foram utilizados apenas os atributos *followers* e *following* de cada usuário, visto que mais de 2 atributos faz com que a visualização não seja ideal para representação gráfica.

O algoritmo foi implementado em Python 2.7 e suas principais funções são as seguintes:

```
def findCloser(user, avgs):
  closest_i = 0
  closest = INF
  for i in range(len(avgs)):
    dist = euclideanDistance(user[1:], avgs[i])
    if dist < closest:
      closest = dist
      closest_i = i
  return closest_i

def newAverages(groups):
  avgs = []
  k = len(groups[0])
  for i in range(len(groups)):
    z = map(list, zip(*groups[i]))
    avgs.append(map(mean, z[1:]))
  return avgs
```

IV. Mapas Auto-Organizáveis

O algoritmo de mapas auto-organizáveis ou self-organizing maps (SOM) possui o mesmo propósito do algoritmo k-Means, entretanto, não possui um número pré-definido de grupos, sendo esses delimitados durante o treinamento. O algoritmo não possui uma convergência tão rapida quanto o k-Means, mas tem uma capacidade de separação maior.

Primeiramente, é definida uma matriz com $n \times m$ pesos inicializados aleatoriamente, sendo m e n as dimensões do mapa. Depois, cada amostra é comparada com todos nodos da matriz, até que um seja definido o nodo mais semelhante à amostra, onde este nodo é chamado de Best Matching Unit (BMU). A partir deste nodo, todos nodos na vizinhança deste serão ajustados de acordo com a seguinte equação:

$$W(t+1) = W(t) + \theta(t) \times L(t) \times (V(t) - W(t)), \quad (2)$$

onde L é o *learning rate* variável com o tempo, representado por

$$L(t) = L_0 \times exp(-\frac{t}{\lambda}) \quad (3)$$

e θ é a influência sobre determinado nodo, baseado na distância ao BMU e representado pela equação

$$\theta(t) = exp(-\frac{dist}{2\sigma(t)}. \quad (4)$$

Nesta equação, $dist$ é a distância euclidiana do nodo analisado ao BMU e σ é o raio da vizinhança do BMU. σ pode ser calculado por

$$\sigma(t) = \sigma_0 \times exp(-\frac{t}{\lambda}), \quad (5)$$

onde t é a iteração e λ é uma constante de tempo. A presença do tempo nesta função garante o decaimento de tanto o learning rate quanto o raio da vizinhança do BMU.

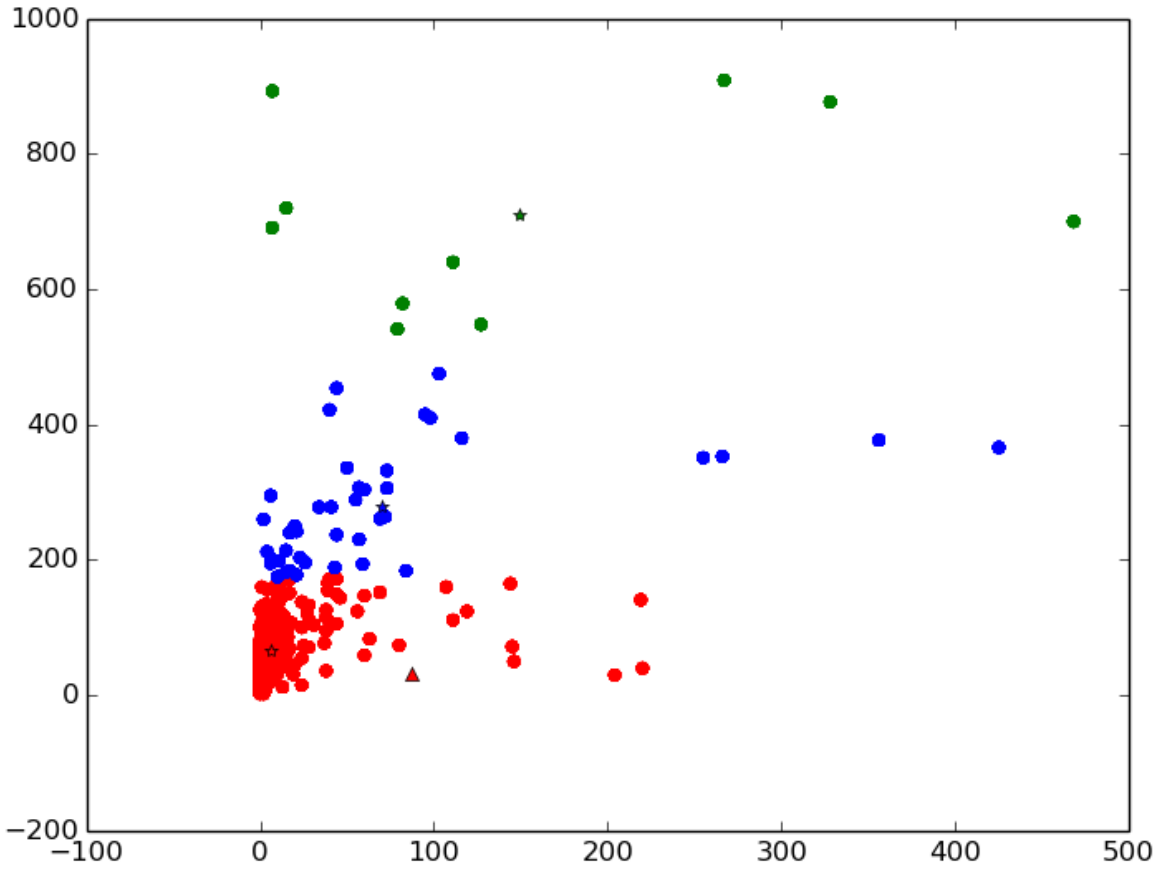

Figura 2. Separação dos grupos no algoritmo k-Means.