MO444 – Aprendizado de Máquina

Edgar Rodolfo Quispe Condori - RA 192617

May 29, 2017

Questão 1. *Os dados quakes.csv representam dodos reais de terremotos (nao sei de que periodo). A primeira coluna é a profundidado do terremoto, a segunda e terceira a latitude e longitude, e a quarta a scala Richer do terremoto. Nos queremos discobrir se ha clusters/grupos de terremotos. Estandarize cada coluna*

```
def preprocess_data():
  #read cvs file
  file_obj = open("quakes.csv", "rt")
  reader = csv.reader(file_obj)
  data = []
  for row in reader:
    data.append(row)
  data = np.array(data).astype(np.float)

  #standardize data
  scaler = StandardScaler().fit(data)
  data = scaler.transform(data)

  return data
```

Questão 2. *Rode o K-means para K=2..10, use random_state=1, e imprima (com 2 casas decimais) a silueta media e o indice de Calinski-Harabaz. Qual parece ser o melhor valor/valores para K?*

Os resultados são:

```
K-means
        Clusters: 2
        Silhouette: 0.32
        calinski Harabaz: 675.01
        Clusters: 3
        Silhouette: 0.35
        calinski Harabaz: 785.77
        Clusters: 4
        Silhouette: 0.34
        calinski Harabaz: 866.69
        Clusters: 5
        Silhouette: 0.34
        calinski Harabaz: 957.38
        Clusters: 6
        Silhouette: 0.36
        calinski Harabaz: 1002.26
        Clusters: 7
        Silhouette: 0.37
        calinski Harabaz: 971.18
        Clusters: 8
        Silhouette: 0.36
        calinski Harabaz: 949.16
        Clusters: 9
        Silhouette: 0.36
        calinski Harabaz: 938.17
        Clusters: 10
        Silhouette: 0.35
        calinski Harabaz: 925.82
```

Baseados no valor da silhueta e calinski Harabaz o melhor número de clusters parece ser 7.

O código:

```
def kmeans_evaluation(data):
    print "K-means"
    for k in range(2,11):
        #k-means clustering model and fit
        kmeans_model = KMeans(n_clusters = k, random_state=1).fit(data)
        labels = kmeans_model.labels_
        print "\tClusters:", k
        print "\tSilhouette: %.2f" % silhouette_score(data, labels, metric='euclidean')
        print "\tcalinski Harabaz: %.2f" % calinski_harabaz_score(data, labels)

if __name__ == '__main__':
    data = preprocess_data()
    kmeans_evaluation(data)
```

Questão 3. *Talvez os dados de latitude e longitude estao estragando a clusterizacao. Remova essa colunas e repita a tarefa acima? Discuta os resultados*

Os resultados são:

```
K-means
        Clusters: 2
        Silhouette: 0.70
        calinski Harabaz: 1483.64
        Clusters: 3
        Silhouette: 0.58
        calinski Harabaz: 3087.43
        Clusters: 4
        Silhouette: 0.51
        calinski Harabaz: 3216.86
        Clusters: 5
        Silhouette: 0.53
        calinski Harabaz: 3125.96
        Clusters: 6
        Silhouette: 0.51
        calinski Harabaz: 3160.03
        Clusters: 7
        Silhouette: 0.49
        calinski Harabaz: 3101.39
        Clusters: 8
        Silhouette: 0.48
        calinski Harabaz: 3150.98
        Clusters: 9
        Silhouette: 0.51
        calinski Harabaz: 3200.48
        Clusters: 10
        Silhouette: 0.53
        calinski Harabaz: 3202.01
```

Comparando esse resultados com os anteriores é facil de ver que o valor da silhueta é melhor para todos os números de clusters. É possibel que isso aconteçe porque é melhor tentar clusterizar os terremotos sem considerar a locação deles, esto faz sentido desde que um terremoto com determinadas caracteristicas seria o mesmo em qualquer parte do mundo. Ao considerar a localização de um terremoto estaria-se tentando identificar tipos de terremotos e classificar zonas pela sua atividade sismica o que torna o problema ainda mais complicado.

Nesse caso o melhor número de clusters é 2.

O código.

```
def kmeans_evaluation(data):
    print "K-means"
    for k in range(2,11):
        #k-means clustering model and fit
        kmeans_model = KMeans(n_clusters = k, random_state=1).fit(data)
        labels = kmeans_model.labels_
        print "\tClusters:", k
        print "\tSilhouette: %.2f" % silhouette_score(data, labels, metric='euclidean')
        print "\tcalinski Harabaz: %.2f" % calinski_harabaz_score(data, labels)
```

```
def remove_lat_lon(data):
    #remove latitute and longitute columns
    return np.hstack((data[:, 0].reshape(-1, 1), data[:, 3].reshape(-1, 1)))

if __name__ == '__main__':
    data = preprocess_data()
    kmeans_evaluation(remove_lat_lon(data))
```

Questão 4. *Rode a clusterizacao hierarquica (metodo Ward) para 2..10 clusters para os dados de 4 dimensoes e calcule os 2 indices acima? Qual o melhor/melhores valores do numero de clusters?*

```
Hirerchical - Ward method
        Clusters: 2
        Silhouette: 0.31
        calinski Harabaz: 536.44
        Clusters: 3
        Silhouette: 0.33
        calinski Harabaz: 698.08
        Clusters: 4
        Silhouette: 0.26
        calinski Harabaz: 731.08
        Clusters: 5
        Silhouette: 0.29
        calinski Harabaz: 771.95
        Clusters: 6
        Silhouette: 0.31
        calinski Harabaz: 750.03
        Clusters: 7
        Silhouette: 0.31
        calinski Harabaz: 741.82
        Clusters: 8
        Silhouette: 0.31
        calinski Harabaz: 733.85
        Clusters: 9
        Silhouette: 0.31
        calinski Harabaz: 745.43
        Clusters: 10
        Silhouette: 0.32
        calinski Harabaz: 749.07
```

Baseados no valor da silhueta e calinski Harabaz o melhor número de clusters é 2 ou 3, isto pode ter alguma relação com que esse methodo é herarquico e o explicado na pergunta 3.

O código:

```
def hierarchical_evaluation(data):
    print "Hirerchical - Ward method"
    for k in range(2,11):
        #hierarchical-means clustering model and fit
        hierarchical = AgglomerativeClustering(n_clusters = k, linkage = "ward").fit(data)
        labels = hierarchical.labels_
        print "\tClusters:", k
        print "\tSilhouette: %.2f" % silhouette_score(data, labels, metric='euclidean')
        print "\tcalinski Harabaz: %.2f" % calinski_harabaz_score(data, labels)
if __name__ == '__main__':
    data = preprocess_data()

    hierarchical_evaluation(data)
```

Questão 5. *Rode o DBScan nos dados de 4 dimensoes. Use 5 como min_samples. Construa o grafico da distancia dos 5-NN e descubra o valor do eps. (Se voce nao conseguir gerar o grafico, use eps = 0.75 mas essa opcao perderá alguns pontos nesta questao). Qual o número de clusters? Calcule os indices acima para os clusteres.*

Para poder identificar um conjunto interessante para o hiperparámetro *eps* foram geradas graficos para cada um dos niveis de vizinhos.

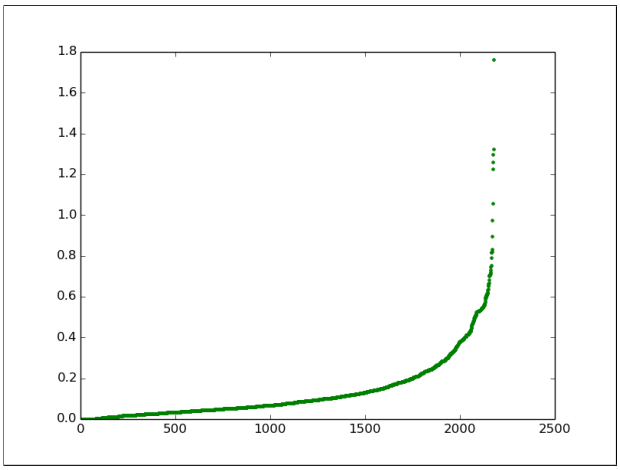

Figure 1: Dintancia dos 1-nn

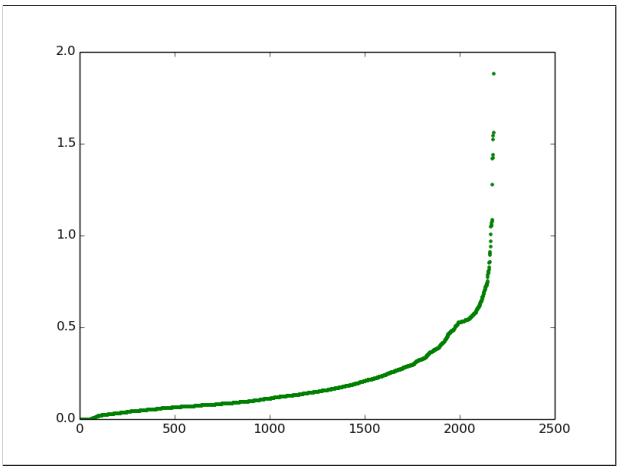

Figure 2: Dintancia dos 2-nn

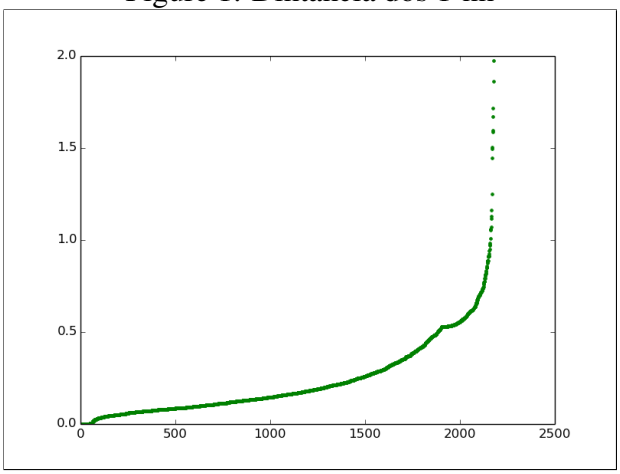

Figure 3: Dintancia dos 3-nn

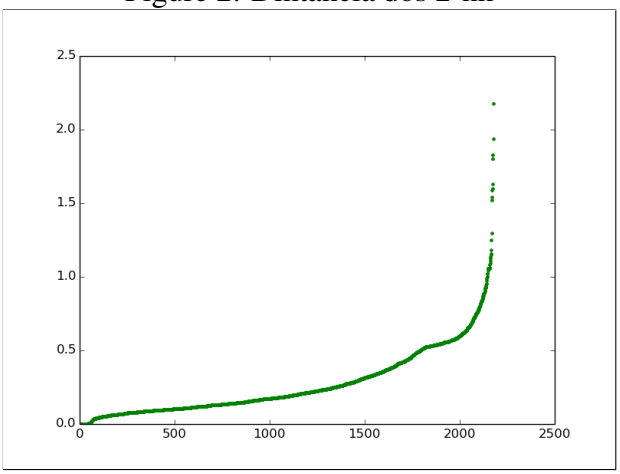

Figure 4: Dintancia dos 4-nn

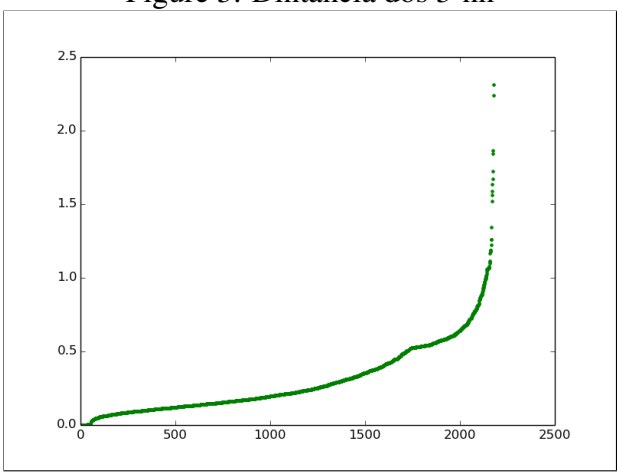

Figure 5: Dintancia dos 5-nn

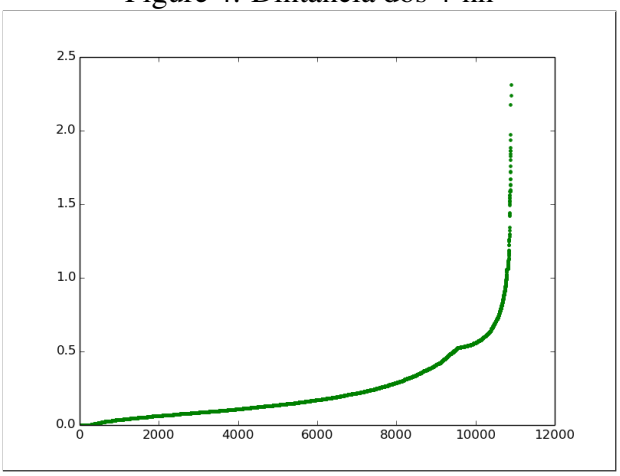

Figure 6: Dintancia dos [1-5]-nn

Note que a construção dessas figuras é baseada no $6-nn$ porque o KNN é ajustado e testado no mismo conjunto de dados, então o primerio vizinho mais perto para cada dado é ele mesmo.

Das figuras é possivel dar un rango de valores para o hiperparametro *eps*, este é: $[0.6, 0.65, 0.7]$, logo os resultados são:

```
DBSCAN
        EPS: 0.6
        Clusters: 10
        Silhouette: -0.00
```

```
calinski Harabaz: 68.25
EPS: 0.65
Clusters: 9
Silhouette: 0.01
calinski Harabaz: 80.23
EPS: 0.7
Clusters: 7
Silhouette: 0.08
calinski Harabaz: 94.44
```

Baseados no valor do calinski Harabaz o valor de $eps = 0.7$ e 7 clusters parece ser a melhor alternativa.

O código.

```
def DBSCAN_evaluation(data):
    print "DBSCAN"
    eps_values = [0.6, 0.65, 0.7]
    for eps_value in eps_values:
        DBSCAN_model = DBSCAN(min_samples = 5, eps = eps_value).fit(data)
        labels = DBSCAN_model.labels_
        # Number of clusters in labels, ignoring noise if present.
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
        print "\tEPS:", eps_value
        print "\tClusters:", k
        print "\tSilhouette: %.2f" % silhouette_score(data, labels, metric='euclidean')
        print "\tcalinski Harabaz: %.2f" % calinski_harabaz_score(data, labels)


def plotDistances(distances, save_file_name = ""):
    plt.figure()
    plt.plot(range(len(distances)), distances, 'g^')
    plt.savefig(save_file_name)

def compute_EPS(data):
    nn = NearestNeighbors(n_neighbors = 6).fit(data)
    distances, indices = nn.kneighbors(data)
    graph = nn.kneighbors_graph(data).toarray()
    print distances[:,1:].mean()
    print distances[:,1:].min()
    print distances[:,1:].max()
    plotDistances(distances[:, 1:].ravel(), "full_dis.png")
    for i in range(1, 6):
        print "====================="
        print i
        print distances[:, i].mean()
        print distances[:, i].min()
        print distances[:, i].max()
        plotDistances(distances[:, i].ravel(), str(i) + "_dis.png")
if __name__ == '__main__':
    data = preprocess_data()
    compute_EPS(data)
    DBSCAN_evaluation(data)
```

Questão 6. *Discuta quantos clusters voce acha apropriado para esse problema.*

Fazendo um resumo dos resultado obtidos, 2 deles indicam que o melhor número de clusters teria ser 2 e os outros dois indican que o melhor número de clustes é 7. Se faz-se um analisis das avordagens dos algoritmos pode se dicer que se precisa-se classificar os terremotos em base a suas caracteristicas (profundidado e scala Richer do terremoto) o numero certo de cluster teria ser 2 mas se a ubicação do terremoto é relevante então o numero certo teria ser 7.