K-means Kümele ile İmgelerin Kümelenmesi

Makine Öğrenmesi 2. Ödevi

Sercan Sözen

Yıldız Teknik Üniversitesi Elektrik-Elektronik Fakültesi Bilgisayar Mühendislği

İstanbul/TÜRKİYE
f0121071@std.yildiz.edu.tr

***Özet*—Bu çalışmada (CALTECH)' ten alınan imgeler öncelikle 32 renge indirgendi. Sonrasında her imgenin her bir renk kanalı için 32 elemanlı 3 adet histogram dizisi oluşturuldu.**

Sonrasında imgelerin histogramları birleştirilerek her bir imgenin sırasıyla "blue, green, red" kanalları için olacak şekilde histogramları bir dizide toplandı. Bu dizi kullanılan 140 adet imgeyi içinde barındırıyor. Her bir eleman bir imgeyi simgeliyor ve her elemeanın 3 alt histogram elemanı var. Bu histogram dizilerinin de her biri 32 eleman uzunluğunda.

Daha sonra bu imgeler arasından rastgele k örnek alınıp imgelerin olduğu diziden silinip kümeleme merkezlerinin bulunduğu "centroids" dizisine eklendi. Sonrasında dizide kalan 140 – k adet elemanın her bir "centroid"e uzaklığı hesaplandı ve imge en yakın olduğu centroidin kümesine eklendi.

Çalışmanın ileri aşamasında başka bir "custom" K-means yaklaşımı ve en sonunda ise Sklearn kütüphanesinin K-means metodu denendi.

Anahtar Kelimeler — makine öğrenmesi, kümeleme, görüntü işleme, histogram

I. GİRİŞ

K-means ve diğer kümeleme (mean shift vb.) algoritmaları etiketlenmemiş verilerin sınıflandırılması için etkili yöntemlerdir. Bu ödevde farklı hayvan ve bitki türlerinin kümelenmesi için kullanılsa da kümeleme algoritmaları sınıflandırılmamış veriler ile çalışılan her alanda kullanılabilir. Örnek olarak sahte haber tespiti, spam filtresi, belli bir görüntüdeki nesnelerin kümelenmesi ve hatta taksi şoförlerinin güzergah sahtekarlığı (Ge, Xiong, Liu, & Zhou, 2011)'nın tespiti için bile kullanılabilir.

II. VERİ KÜMESİ VE ÖNİŞLEMLER

Çalışmada kullanılan veri seti (CALTECH)'ten alınmıştır. 7 ayrı kategorideki (octopus, elephant, flamingo, kangaroo, leopards, sea_horse, sunflower) imgelerden ilk 20 tanesi seçilmiş ve ortak bir veri seti oluşturulmuştur.

A. Verilerin İçe Aktarılması

Veri setindeki tüm imgeler glob kütüphanesi kullanılarak tek bir listeye import edilmiştir.

```
images = [cv2.imread(file) for file in glob.glob("all/*jpg")]
```

***Şekil 1** Verilerin içe aktarılması*

B. İmgelerin 32 Renge İndirgenmesi

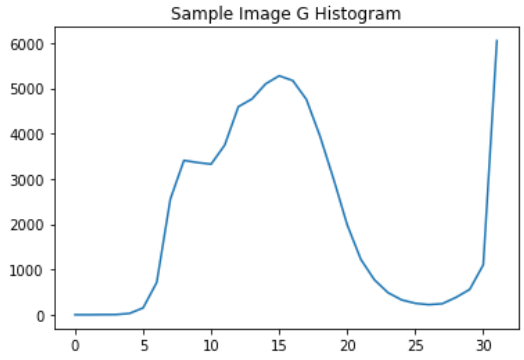
Veri setindeki imgelerin üstünde yapılacak işlemler için kolaylık sağlamak amacıyla yeterli sayıda renk çeşidi kullanmak yararlı olacaktır. Bu sebeple imgelerin tümü hazırlanan "quant" fonksiyonuyla k-means kullanılarak 32 renge indirgendi.

```
def quant(img,k):
    data = np.float32(img).reshape(-1,3)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,20,1.0)
    ret,label,center = cv2.kmeans(data, k , None, criteria,10,cv2.KMEANS_RANDOM_CENTERS)
    center = np.uint8(center)
    result = center[label.flatten()]
    result = result.reshape(img.shape)
    return result

for n in range (0,140):
    images[n] = quant(images[n],n_colors)
```

***Şekil 2** İmgelerin 32 renge indirgenmesi*

C. Renk Histogramlarının Çıkarımı

İmgelerin renk histogramlarını tutması için 3 kanalın her birinin histogram değerlerinin tutulacağı 3 ayrı dizi oluşturuldu. OpenCV kütüphanesinin "calcHist" fonksiyonu kullanılarak her bir renk kanalının histogramları çıkarıldı. *Şekil 1*' de örnek bir resmin yeşil renk histogramını gösterilmiştir.



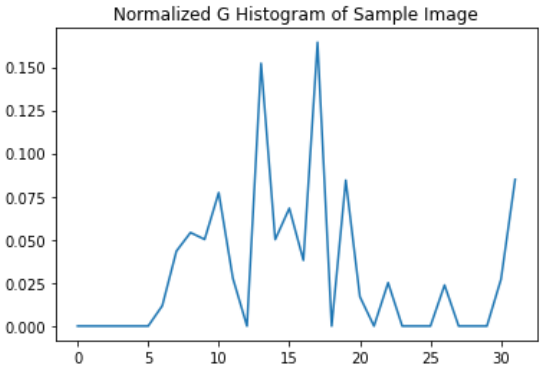
***Şekil 3** Rastgele bir imgenin yeşil renk histogramı*

D. Histogram Değerlerinin Normalizasyonu

Her resimde, her renk bileşeni için histogram dizisindeki sonuçlar, toplam piksel sayısına bölünerek dizi elemanlarının değeri [0-1] aralığına normalize edildi. Şekil normalize edilmiş histogramı göstermektedir.

```
for i in range(0,140):
    img_pixels = images[i].size/3
    bhists[i] = bhists[i] / (img_pixels)
    ghists[i] = ghists[i] /(img_pixels)
    rhists[i] = rhists[i] /(img_pixels)
```

***Şekil 4** Renk değerlerinin normalizasyonu*



Şekil 5 *Normalize edilmiş histogram*

III. SİSTEM TASARIMI VE K-MEANS ALGORİTMALARI

A. Özelleştirilmiş K-means 1

Bu ödev için 2 farklı özel k-means algoritması denedim. Sonrasında da sklearn kütüphanesini kullandım. İlk olarak denediğim algoritmada öncelikle tüm renk kanalları için histogramlar bir dizide toplandı ve her biri imgelere dağıtıldı. Sonuçta ortaya çıkan 140 elemanlı dizide her bir eleman bir imgeyi simgelerken, her elemanın da renk kanallarının histogramlarını temsil eden 3 alt elemanı var. Bu 3 alt elemanın da her birinin 32 elemanı var ve bu elemanlar histogramdaki değerleri temsil ediyor.

Daha sonra oluşturulan bu diziden rastgele k adet imge Şekil 6‘ daki gibi kümeleme merkezi (centroid) olarak seçilip ana diziden siliniyor.

```
import random
k = 7 # of centroids

all_img = list(zip(bhists, ghists, rhists)) #all images' histograms with order.

centroids = []
for i in range (k):
    a = random.randint(0, 139)
    centroids.append(all_img[a])
    del all_img[a]

print(len(all_img))
print(len(centroids))
```

Şekil 6 *Kümeleme merkezlerinin çıkarılması*

Sonrasında ana dizideki her bir elemanın centroid’lere olan uzaklıkları Şekil 7‘ deki fonksiyon ile hesaplanıyor ve imge en yakın olduğu centroid’in kümesine ekleniyor.

```
def get_distance(p, q):
    """
    aynı boyutlu
    p ve q noktalarının arasındaki öklid uzaklık
    """
    # sum of squared difference between coordinates
    s_sq_difference = 0
    for p_i,q_i in zip(p,q):
        s_sq_difference += (p_i - q_i)**2

    # take sq root of sum of squared difference
    distance = s_sq_difference**0.5
    return distance
```

Şekil 7 *Öklid uzaklık hesaplanması*

Buradaki öklid fonksiyonunun kullanılabilmesi için centroidlerin ve imgelerin bulunduğu listelerin sırasıyla (140-k, 3, 32, 1) ve (k, 3, 32, 1) olan boyutları, listeler numpy dizilerine çevrilip her ikisi de 2x2’ lik birer dizi haline getirildi.

```
X = np.array(all_img)
print(X.shape)
X = X.reshape(len(all_img),3*32*1)

C = np.array(centroids)
print(C.shape)
C = C.reshape(k,3*32*1)
print(C.shape)
```

Şekil 8 *Ana dizi ve centroid dizinin yeniden boyutlandırılması*

Yeniden boyutlandırılan diziler şekil2 deki döngü kullanılarak her bir imgenin her bir centroid’e mesafesi hesaplandıktan sonra imgenin en yakın mesafede olduğu centroid’in index değerine bakılarak imgeler yeni bir dizide atandı. Sonuçta elde ettiğimiz bu diziler kümeleme sonuçlarımızı ifade etmektedir.

```
for i in range(len(all_img)):
    dist = []
    for j in range(k):
        d = get_distance(X[i], centroids[j])
        a = sum(d) / len(d)
        dist.append(a)
    if (dist.index(min(dist)) == 0):
        cluster0.append(X[i])

    elif (dist.index(min(dist)) == 1):
        cluster1.append(X[i])

    elif (dist.index(min(dist)) == 2):
        cluster2.append(X[i])

    elif (dist.index(min(dist)) == 3):
        cluster3.append(X[i])

    elif (dist.index(min(dist)) == 4):
        cluster4.append(X[i])

    elif (dist.index(min(dist)) == 5):
        cluster5.append(X[i])

    elif (dist.index(min(dist)) == 6):
        cluster6.append(X[i])
```

Şekil 9 *Kümeleme döngüsü*

```
Cluster0 elements count:  0
Cluster1 elements count:  98
Cluster2 elements count:  16
Cluster3 elements count:  14
Cluster4 elements count:  5
Cluster5 elements count:  0
Cluster6 elements count:  0
```

Şekil 10 *Kümeleme sonuçları örneği*

B. Özelleştirilmiş K-means 2

İkinci olarak kullandığım kümeleme algoritması ise parametre olarak k değeri, tolerans değeri (sklearn’deki k-means’e benzer) ve maksimum iterasyon sayısı (max_iter) almaktadır. Veri setinden k kadar centroid seçilir ve sonrasında her bir elemanın centroid’e uzaklığı hesaplanır. Sonraki iterasyonlarda önceki iterasyondan elde edilen kümelerin yeni merkezleri alınarak yeni seçilen centroid’lerden farkına bakılır. Bu fark tolerans değerinden küçük ise bu kümeleme optimize olarak kabul edilir. Eğer

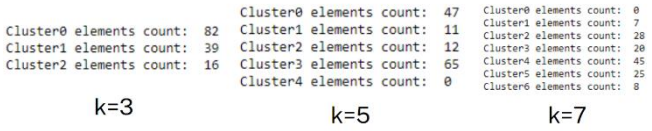
```
optimized = True

for c in self.centroids:  # centroidi tolerans ile yenile
    original_centroid = prev_centroids[c]
    current_centroid = self.centroids[c]
    if np.sum((current_centroid-original_centroid)/original_centroid*100.0) > self.tol:
        print(np.sum((current_centroid-original_centroid)/original_centroid*100.0))
        optimized = False

if optimized:
    break
```

Şekil 11 *Tolerans değeriyle yeni centroid'lerin belirlenmesi*

tolerans değerinden büyük ise iterasyon devam eder.

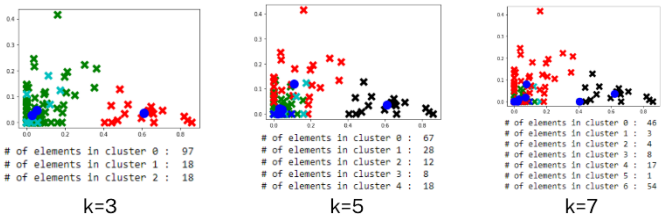
IV. DENEYSEL ANALİZ

Algoritma 1’in elde ettiği sonuçlara önceki bölümdeki Şekil 10’da değinilmişti. Farklı k değerleri için Algoritma 1’in kümeleme sonuçları ise Şekil 12’deki gibidir.



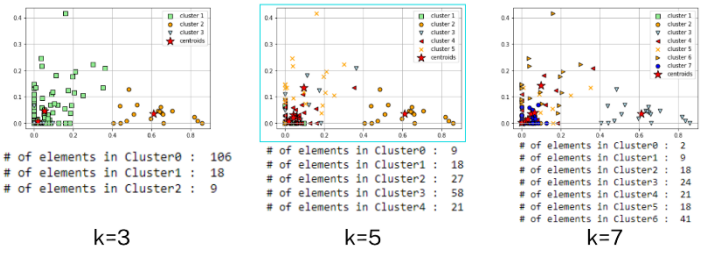
Şekil 12 *Algoritma 1'in k=3, 5 ve 7 için kümeleme sonuçları*

Buna ek olarak Algoritma 2'nin elde ettiği kümeleme sonuçları ise k=3, 5 ve 7 için Şekil 13'teki gibi gerçekleşti. Şekil 13'te mavi noktalar küme merkezlerini temsil etmektedir.



Şekil 13 *Algoritma 2'nin k=3, 5 ve 7 için kümeleme sonuçları*

Daha sonra Sklearn kütüphanesindeki K-means'i kullanarak bir deneme yapıldı.



Şekil 14 *Sklearn K-means kümeleme sonuçları*

V. REFERANSLAR

CALTECH, C. V. (2011). Caltech 101 Dataset.

Ge, Y., Xiong, H., Liu, C., & Zhou, Z.-H. (2011). A Taxi Driving Fraud Detection System. *2011 IEEE 11th International Conference on Data Mining*.