

Gaussian Karisimleri ile Deri Rengi Saptamak

Bir projemizde dijital resimlerdeki deri rengi içeren kısımları çıkartmamız gerekiyordu; çünkü fotoğrafın diğer renkleri ile ilgileniyorduk (resimdeki kişinin üzerindeki kıyafetin renkleri) ve bu sebeple deri renklerini ve o bölgeleri resimde saptamak gerekti. Bizim de önceden aklımızda kalan bir tembih vardı, Columbia Üniversitesi'nde yapay öğrenim dersi veren Tony Jebara derste paylaştı bir kere (bu tür gayri resmi, lakirdi seviyesinde tiyolar bazen çok faydalı olur), deri rengi bulmak için bir projesinde tüm deri renklerini R,G,B olarak grafiğe basmışlar, ve beyaz olsun, zenci olsun, ve sonuç grafikte deri renklerinin çok ince bir bölgede yanyana durduğunu görmüşler. İlginç değil mi?

Buradan şu sonuç çıkıyor ki diğer renklerin arasında deri renklerine odaklanan, onları "taniyan" bir yapay öğrenim algoritmasının oldukça şansı vardır. Ama ondan önce veriyi bakıp grafiksel olarak ne olduğunu görelim.

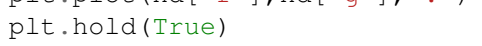
```
import pandas as pd, zipfile
with zipfile.ZipFile('skin.zip', 'r') as z:
    d = pd.read_csv(z.open('skin.csv'), sep=',')
print d[:3]
```

	Unnamed: 0	rgbhex	skin	r	g	b	h	\
0	0	#200e08	False	0.125490	0.054902	0.031373	0.041667	
1	1	#6d6565	False	0.427451	0.396078	0.396078	0.000000	
2	2	#1f2c4d	False	0.121569	0.172549	0.301961	0.619565	

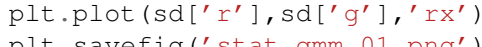
	s	v
0	0.750000	0.125490
1	0.073394	0.427451
2	0.597403	0.301961

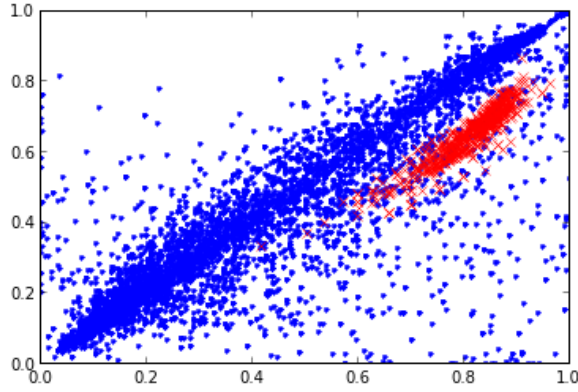
Burada önemli olan R,G,B ve H,S,V kolonları. Bu iki grup değişik renk kodlama yöntemini temsil ediyorlar. Grafikleyelim,

```
nd = d[d['skin'] == False]
sd = d[d['skin'] == True]
plt.plot(nd['r'], nd['g'], '.')
```



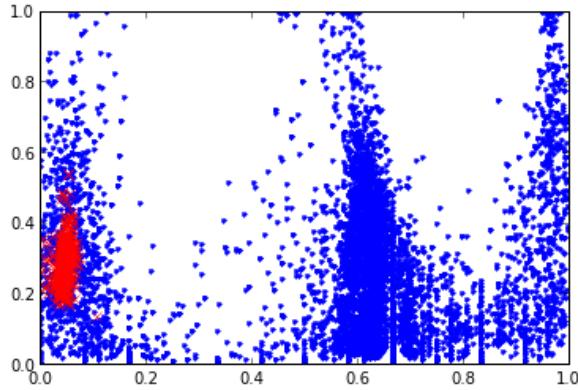
```
plt.hold(True)
plt.plot(sd['r'], sd['g'], 'rx')
plt.savefig('stat_gmm_01.png')
```





Ya da H,S uzerinden

```
nd = d[d['skin'] == False]
sd = d[d['skin'] == True]
plt.plot(nd['h'], nd['s'], '.')
plt.hold(True)
plt.plot(sd['h'], sd['s'], 'rx')
plt.savefig('stat_gmm_02.png')
```



Demek ki Jebara hakliyimis. Veriye bakınca bir kabaca / sezgisel (intuitive) bazı çıkarımlar yapmak mümkün. Mesela her iki grafikte de deri renklerini belirten bolgenin grafiği sanki 3 boyutlu bir Gaussian'ın üstten görünen / kontur (contour) hali. Bunu bilmek bir avantaj, bu avantajı kullanmak lazım. Eğer modelimiz gerçek dünya verisine ne kadar yakınsa, yapay öğrenim sansi o kadar fazlalaşacaktır. Eğer o bölgeye bir Gaussian uydurursak (fit) tanıma sansimiz artacaktır.

O zaman deri rengi tanıma şu şekilde yapılabilir. Scikit Learn kutuphanesinin Gaussian Karışımları (GMM) içeren çok güzel bir paketi var, bunu kullanabiliriz. Tek problem bu karışımlar olasılık fonksiyonunu öğreniyorlar, sınıflama (classification) yapmıyorlar. Önemli değil, şöyle bir ek kod ile bunu halledebiliriz; iki tane GMM yaratırız, bir tanesi deri renk bölgeleri için, diğeri diğer bölgeler

icin. Egitim sirasinda her iki GMM'i kendi bolgeleri uzerinde egitiriz. Sonra, test zamaninda, her yeni (bilinmeyen) veri noktasini her iki GMM'e veririz, hangisinden daha yuksek olasilik degeri geliyorsa, etiket degeri olarak o GMM'in degerini aliriz.

GMM'leri, ve onların icindeki Gaussian'ların kovaryanslarını kullanmak faydalı, kovaryans bildigimiz gibi bir Gaussian'ın hangi yonde daha fazla agirliginin olacagini belirler, eger kovaryans hesabi yapılmazsa, yani kovaryans matrisinin sadece caprazinda degerler varsa, mesela uc boyutta Gaussian'ın konturu bir cember olarak gozukur [1, sf 90]. Tabii her yonde ayni agirlikta olan bir Gaussian her türlü veriyi temsil edemez, en esnegi (ki grafige bakınca bu gerekliligi goruyoruz) tam kovaryans kullanmaktır. Scikit Learn ile bu secim GMM için `full` ile yapilir, sadece caprazi kullan anlamına gelen `diag` da olabilirdi.

```
import zipfile
from sklearn.cross_validation import train_test_split
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn.mixture import GMM
import pandas as pd

class GMMClassifier():
    def __init__(self, k, var):
        self.clfs = [GMM(n_components=k,
                          covariance_type=var, thresh=0.1,
                          min_covar=0.0001, n_iter=100) for i in range(2)]

    def fit(self, X, y):
        self.clfs[0].fit(X[y==0])
        self.clfs[1].fit(X[y==1])

    def predict(self, X):
        res0 = self.clfs[0].score(X)
        res1 = self.clfs[1].score(X)
        res = (res1 > res0)
        return res.astype(float)

if __name__ == "__main__":
    with zipfile.ZipFile('skin.zip', 'r') as z:
        df = pd.read_csv(z.open('skin.csv'), sep=',')
        y = (df['skin'] == True).astype(float)
        X = df[['h', 's', 'v', 'r', 'g']]

    res = []
    for i in range(5):
        clf = GMMClassifier(k=10, var='full')
        x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=2000)
        clf.fit(x_train, y_train)
        preds = clf.predict(x_test)

        fpr, tpr, thresholds = roc_curve(y_test, preds)
        roc_auc = auc(fpr, tpr)
```

```
res.append(roc_auc)

print 'deneyler', res
print 'nihai ortalama', np.array(res).mean()

deneyler [0.99075081610446136, 0.98417442945172173, 0.98641291695170819,
          0.98779826464208242, 0.99239130434782608]
nihai ortalama 0.9883055463
```

Basari oranı yüzde 98.8! Bu problem üzerinde pek çok diğer yöntem denedik, mesela KNN sınıflayıcı, Lojistik Regresyon, vs. gibi, bu yöntem tüm diğerlerini geçti.

İlginc bir yan bir soru, “hangi kolonların kullanılacağı”. Bu bağlamda projede arkadaşlardan “ama HSV değerleri RGB değerlerinden türetilbiliyor, ya birini ya ötekini kullanmak yeterli olmaz mı?” yorumu yapanlar oldu. Evet, bu verinin diğerinden “türetilmiş” olduğu doğru, ve beklenir ki ideal bir dünyada mükemmel bir yapay öğrenim algoritmasının bu tür bir yardıma ihtiyacı olmaz, algoritma o kadar iyidir ki ona sanki aynı veriyi tekrar vermiş gibi oluruz, en iyi ihtimalle ek kulfet yaratırız. Fakat pratikte bu ek veri algoritmaya ek bazı sinyaller verebilir. Mesela eğer müşterilerin kilosu üzerinden bir öğrenim yapıyor olsaydık, 80 kilodan daha az ya da daha fazla olmayı (problem alanına göre) ayrı bir kolon olarak kodlamak avantaj getirebilirdi. Tabii ki kilo verisi numerik değer olarak ağırlıkla fazlasıyla oradadır, fakat önem verdiğimiz noktaları türetilmiş veri olarak öğrenim algoritmasına vermenin zararı yoktur. Üstteki örnekte GB değerlerinin HSV ile beraber kullanılmasının başarı şansını biraz daha arttırdığını görebiliriz.

Kaynaklar

- [1] Alpaydin, E., *Introduction to Machine Learning*
- [2] Jebara, T., *Columbia Machine Learning Course*