Lec5_Image Retrieval

Clustering(Unsupervised Learning): maximize the inter-cluster distance while to minimize the intra-cluster distance.

K Means

1. Pick a number (K) of cluster centers (at random)
2. Assign every item to its nearest cluster center (e.g. Euclidean distance)
3. Move each cluster center to mean of its assigned items
4. Repeat 2,3 until convergence (change in cluster assigned less than a threshold)

Content based image retrieval (CBIR) - one feature per image

looking for img composed of similar content

1. Extract features vectors of all images on file
2. Extract feature vector for the query image
3. Compare it to all (target) images on file by calculating query target similarities
4. Sort similarities in a descending order with ranked list of targets

Better way:

Indexing: group img as clusters and pick one from each cluster as its representative(tree) Coarse : compare to representatives only and find top-k Fine : compare to the member images of the top-k clusters

Bag of Visual Words (BoVW) - multiple(local) feature vectors

1. Visual Descriptor Extraction
2. Dictionary(CodeBook) - clustering, pick 1 from cluster, put them together to construct dict
3. Count words in a bag - calc frequency in bag & construct histogram as feature vector

- Text-based: keyword, description, annotation
- sematic-based: relevance feadback, automatic

Tutorial

```
**K-means**
flags = cv2.KMEANS_PP_CENTERS
# TODO: use cv2.kemeans to cluter the images
compactness, labels, centers = cv2.kmeans(np.float32(np.stack(features, axis=0)),
4, None, criteria, 10, flags)

**t-SNE**
tsne = TSNE()
X_embedded = tsne.fit_transform(features)
sns.scatterplot(x=X_embedded[:,0], y=X_embedded[:,1], ...)

**CBIR**
```

```
# feature point -> cal distance distance/similarities to other point, sort result
& rank
# Calculate the distance between images
dists = np.linalg.norm(features - f, axis=1)
# Rank the distance and get the first 4 most closest image
ids = np.argsort(dists)[1:5]

**BoVW**
from scipy.cluster.vq import *
from sklearn import preprocessing
# The idea is to model each image as a "bag" with visual words (representative
descriptors) inside.
# By counting the freq of visual words, we can have a histogram as the feature
vector.

# **Step 1. Extract Local Discriptor**
kpt, des = sift.detectAndCompute(img_gray, None) # get SIFT descriptors
des_list.append((fname,des))

# **Step 2: Pool all descriptors, clustering, build dict**
# Stack all the descriptors vertically in a numpy array
descriptors = des_list[0][1]
for image_path, descriptor in des_list[1:]:
    descriptors = np.vstack((descriptors, descriptor))
# Perform k-means clustering (using scipy.cluster.vq.kmeans)
print("Start k-means: %d words, %d key points" %(numWords, descriptors.shape[0]))
voc, variance = kmeans(descriptors, numWords, 1)

**# Step 3: Counting words in each bag (image) to build the histograms**
# build histograms
im_features = np.zeros((len(imgs), numWords), "float32")
for i in range(len(imgs)):
    # TODO: using scipy.cluster.vq
    words, distance = vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w] += 1
# perform L2 normalization
im_features = preprocessing.normalize(im_features, norm='l2')

# **step 4: search with the features and draw the results**
# get score matrix for all searched images
score = np.dot(im_features, im_features.T)
# get ranked list
rank_ID = np.argsort(-score)
```