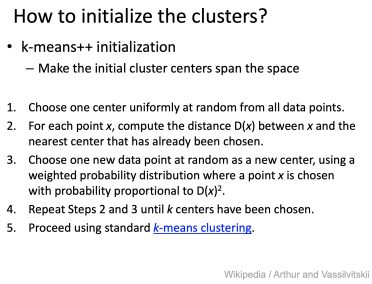
Project 3 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side.
- **Please make this document anonymous.**

In the beginning...

Tiny images: I use skimage.transform.resize() to downsize the image to a 16-by-16 scale and then flatten the ndarray.
Nearest Neighbor: I compute the Euclidean distance between testing features and training features (using cdist()) and label the testing image with its nearest neighbor.
Building Vocabulary: First, I extract features from each image using hog() with the following parameters:
cells_per_block = (4,4)
pixels_per_cell = (16,16)
I tested several different parameters such as (cells_per_block = (4,4), pixels_per_cell = (8,8)), (cells_per_block = (2,2), pixels_per_cell = (8,8)), (cells_per_block = (2,2), pixels_per_cell = (16,16)), etc. Every time I tried a new set of parameters, I saved the corresponding features extracted. Most of them resulted in a huge file (more than 500MB) and therefore, too many features were extracted. So, I ended up with the parameters shown above.
I initialized the centers by the algorithm taught in class.



```
features = np.load('features.npy')
l_f = len(features)

print("Loaded all features")
print("Start to load or initialize k centers")
if not os.path.isfile('incenters.npy'):
    print("No previous saved initialized centers")
    print("Now, initializing k centers")
    #initialize k centers#
    centers_ind = set()

    if l_f<vocab_size:
        print("error, vocab_size too large")
        return np.array([])

    centers_ind.add(np.random.randint(l_f))
    count_center = 1
    for i in range(1,vocab_size):
        sq_dists = np.square(cdist(features, features[list(
                                            centers_ind)], '
                                            euclidean'))
        sum_dists = np.sum(sq_dists,axis = 1)
        dis = sum_dists/np.sum(sum_dists)
        ind = np.random.choice(l_f, p = dis)
        while ind in centers_ind:
            ind = np.random.choice(l_f, p = dis)
        centers_ind.add(ind)
        count_center+=1
        if i % 25 ==0:
            print("current progress:", i)
    centers = features[list(centers_ind)]
    np.save('incenters.npy',centers)
    print("Done initializing k centers")

centers = np.load('incenters.npy')
```

Then, using the standard k-clustering algorithm shown as below, build a dictionary:

```
old_assignments = np.zeros(len(features))-1

safe_counter = 0
while True:
    if safe_counter >100:
        break
    clustering = [[] for i in range(vocab_size)]


    x_dists = cdist(features, centers, 'euclidean')
    new_assignments = np.zeros(l_f)
    for j in range(len(x_dists)):
        ind = np.argmin(x_dists[j])
        new_assignments[j] = ind
        clustering[ind].append(features[j])

    centers = []
```

```
        for i in range(vocab_size):
            centers.append(np.mean(clustering[i],axis = 0))

        if np.sum(new_assignments==old_assignments) >= (0.99*l_f):
            break
        if safe_counter % 5==0:
            print("k-clustering progress:", safe_counter)
        safe_counter += 1
        old_assignments = new_assignments
```

Bags of Words: After building the vocabulary, the bag of words function is just computing histograms of matched vocabularies of every image (also using hog()).
Linear SVM: SVM using LinearSVC with tol=0.001.

A Result

1. Tiny images with Nearest neighbor:
   The accuracy is 18.7%.
2. Bags of words with Nearest neighbor:
   The accuracy is 47.3%.
3. Bags of words with Linear SVM:
   The accuracy is 56.0%.