CS 231A Computer Vision, Spring 2016
Problem Set 4

Due Date: May 20th 2016 11:59 pm

1 Viewpoint Estimation With Bag of Words

In the previous problem set, we explored object detection with histogram of oriented gradients and support vector machines. Often, we not only care about detecting the object, but also recognizing its pose. In this problem, we will build a classifier for estimating one of 16 orientations of different cars, using a BOW method introduced with the EPFL car dataset [1].

Download the starter code from `http://cs231a.stanford.edu/hw/ps4/viewpoint.zip`

Again, download vlfeat: `http://www.vlfeat.org/download/vlfeat-0.9.20-bin.tar.gz`. Put this in a `vlfeat` directory inside the project root folder.

If you ever want more detail or get stuck, referring to lecture notes and the CA section will help.

(a) **[10 points]** The first step is to build a dictionary of codewords which we will use to construct our Bag of Words models. To do so, run kmeans to find the centers of clusters of dense SIFT features in `generateClusterCenters.m`. Submit your code.

(b) **[20 points]** The next step is to now build our bag of words per image. To do so, we simply compute a histogram of codewords occurencies in the image as discussed in lecture. We then repeat this using a spatial pyramid of different grid sizes. Implement this in `generateBOWFeatures.m`. As a sanity check, using a subset of the training data, you should, on most runs, get above 20% classification accuracy. Submit your code.

(c) **[5 points]** Now train and run the SVM classifier using all the training examples in `viewpointEstimation.m`. Because we're working with a lot of images, be prepared for this to take some time. You should see an accuracy above 60% on the test set, well above random chance ($\frac{1}{16}$). Submit your train and test accuracies.

(d) **[5 points]** Let's explore the properties of this Bag of Words model. What happens if we don't use a spatial pyramid (set `pyramid_depth=1`) to augment our feature set? What happens as we decrease the number of words (set `num_centers=2`) in our Bag of Words? Explain your results.

[1] `http://cvlabwww.epfl.ch/~lepetit/papers/ozuysal_cvpr09.pdf`

2 Image Segmentation

Image segmentation is the process of partitioning an image into meaningful structures. Two famous segmentation methods are meanshift [2] and normalized cut [3]. In this problem, you will implement significant parts of both methods.

Download the starter code from http://cs231a.stanford.edu/hw/ps4/segmentation.zip

If you ever want more detail or get stuck, referring to lecture notes and the CA section will help.

Meanshift

(a) **[25 points]** Complete image segmentation using meanshift in `meanshift.m`. Submit your code and the resulting segmentation. View your segmentation results using `segmentation.m`.

(b) **[5 points]** Try the different images provided (`plates.jpg` and `rocks.jpg`). Play around with the bandwidth parameter to get good results on these images and explain the effect of changing it. Also provide your segmentations on these images (try to get the best you can, but don't spend too much time).

Normalized cut

Below, we will both discuss the inspiration for and outline the major steps in segmenting an image by normalized cuts. Feel free to read the paper for the complete mathematical derivation.

The idea for normalized cuts stems from graph theory, where we treat the image as a connected graph $G = (V, E, W)$, with V being the node set, E being the edge set, and W being the matrix of weights for each edge. In this case, the pixels are the nodes and weighted edges are some function of the pixel features. The general min-cut problem assumes we have some node set $V = \{1, ..., n\}$ that we want to partition into subsets A and B for clustering purposes. For segmentation, a minimum cut generally does not work, because it will usually divide A and B into subsets with 1 node and $n - 1$ nodes respectively, which is usually a suboptimal clustering scheme. On the other hand, normalized cuts compute the cut cost as a fraction of each subgroup's total edge connections to the rest of the nodes in the graph. Therefore, normalized cuts favor a more evenly distributed partitioning when minimized.

We are given W, a weight matrix where w_{ij} is the weight of the edge between nodes i and j, and we compute D, the diagonal matrix where $D_{i,i} = \sum_j w_{ij}$. Let x be a vector where $x_i = 1$ if $i \in A$ and $x_i = -1$ if $i \in B$. Then we define

$$k = \frac{\sum_{x_i > 0} D_{i,i}}{\sum_i D_{i,i}}.$$

Intuitively, k is the sum of weights of all the edges in A over the total weight of the graph. Next we define

$$b = \frac{k}{1 - k},$$

[2] https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf
[3] http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf

which is the proportion of the weights of A versus the weights of B. We use this to define

$$y = (1 + x) - b(1 - x),$$

which allows us to set up the facts that

$$y^T D1 = 0 \text{ and } y^T Dy = b1^T D1.$$

Using the previous defintions, we can derive that

$$\min_x \text{ncut}(x, W, D) = \min_y \frac{y^T(W - D)y}{y^T Dy}$$

subject to the constraints

$$y_i \in \{1, -b\},\ y^T D1 = 0.$$

This formulation gives us a simpler way to find the partition that minimizes the normalized cut. By solving for y, which ultimately is just a reformulation of our partition indicator vector x, we are able retrieve the optimal partition A, B. We approximately solve for y by solving the eigensystem

$$(D - W)y = \lambda Dy,$$

which automatically satisfies the constraint $y^T D1 = 0$. The paper finds that the eigenvector v associated with the second smallest (in absolute value) eigenvalue of the generalized eigensystem is the approximate solution to the minimum normalized cut problem. However, we find that it does not necessarily satisfy the other constraint $v_i = \{1, -b\}$. In the ideal case, this eigenvector takes on the two values, which makes partitioning the nodes easy. However, this eigenvector will generally take on continuous values and we need to find a splitting point ϵ such that all indices i such that $v_i > \epsilon$ are in A and otherwise in B. One can take conjectures at the value of the splitting point, such as 0 or the median, but searching for the optimal point works by far the best (use `fminsearch`). Thus, after finding the splitting point, we can compute x, k, b, and y to find the value of minimum normalized cut and its resulting partition.

(a) **[10 points]** Fill in the normalized cut computation in `ncut.m`.
Recall that ncut $= \frac{y^T(D-W)y}{y^T Dy}$. We can compute y from the second smallest eigenvector v. By defining a partition value ϵ, we find that node $i \in A$ if $v_i > \epsilon$, otherwise $i \in B$. Submit your code.

(b) **[10 points]** Usually, a multi-segment segmentation is of interest, rather than a simple, bimodal segmentation (multiple objects or regions of interest in the image). Thus, we recursively repeat it on each subgroup A,B, after each cut until some criteria are satisfied. The overall normalized cut for segmentation is as follows:

1. Given a set of features, set up a weighted graph $G = (V, E, W)$, compute the weight on each edge, and summarize the information into W and D (we provide this code).
2. Solve $(D - W)x = \lambda Dx$ for eigenvectors with the smallest eigenvalues. You will find the matlab function call `eigs(D-W,D,2,'sm')`, which returns the two smallest (in absolute value) eigenvectors that solve this eigensystem, to be extremely useful.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph by finding the partition point ϵ such that ncut is minimized. You might find the Matlab function `fminsearch` useful (initialize the partition point to be 0).

4. Decide if the current partition should be subdivided by ensuring that the ncut is below the threshold value (otherwise we will be partitioning regions that should belong to the same segment). Also check if the each subgraph is large - whether its size is greater than some lower bound pixel count.
5. Recursively repartition the segmented parts if necessary.

Implement this in `recursivePartition.m`. Submit your code.

(c) **[5 points]** View your segmentation results using `segmentation.m`. Submit the resulting segmentation.

(d) **[5 points]** Try the different images provided (`plates.jpg` and `rocks.jpg`). Play around with the ncut and pixel thresholds to get good segmentations of each and explain the effects of changing each. Also provide your segmentations on these images (try to get the best you can, but don't spend too much time).