Gmail by Google

Peter Sung <petersungjfriend08@gmail.com>

[CML, Fall 2015]: Sec. 3 in homework #5, project-specific instructions

1 message

Zaid Harchaoui <zaid.harchaoui@nyu.edu> Wed, Nov 18, 2015 at 11:32 AM
To: Zaid Harchaoui <zaid.harchaoui@nyu.edu>
Cc: Martha Eleanor Poole <mep505@nyu.edu>, Mohammad Afshar <ma2510@nyu.edu>, Peter Sung <yss265@nyu.edu>, Hung-Ting Wen <htw230@nyu.edu>, Mayank Singh <ms8599@nyu.edu>

Dear all,

Please find below the instructions to instantiate Sec. 3 of Homework #5 to your project.

I very strongly recommend to read this now to organize and anticipate the time required for your homework. Experiments take time to run. If you launch experiments in a smart way, those computers can do a lot of work in the background, while you can work on something else. You can anticipate the time required, and schedule your experiments to run on HPC/CIMS clusters

1. Online Mini-batch k-means

The goal here is to write a stochastic-gradient-type extension of your k-means algorithms that can process large datasets. This will be useful to compute the vocabulary for your project from a large number of windows/patches.

Write a function online-kmeans that implements Algo. 1 in [http://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf]. You are can initialize this function with the output of your k-mean algorithm on a small enough sub-sample. Read sklearn.cluster.MiniBatchKMeans, and use speeding-up tricks to accelerate your own implementation. Run comparisons between online-kmeans and my-kmeans++ on synthetic data of increasing size, similar to what you've done for SGD and gradient descent. Also, similar to Sec. 2 of the homework, extract from experiments on synthetic data a rule-of-thumb that automatically chooses between online-kmeans and my-kmeans++, depending on n, d, k (the number of clusters) to get the centroids in a timely manner. Now, the function my-kmeans automatically chooses between the two algorithm based on this rule-of-thumb.

2. VLAD

The goal here is to write a code that implements the VLAD feature representation for your project.

Reference papers for VLAD:

[a] https://courses.cs.washington.edu/courses/cse590v/13au/arandjelovic13.pdf
[b] https://lear.inrialpes.fr/pubs/2010/JDSP10/jegou_compactimagerepresentation.pdf

Refresh on VLAD by reading these two papers. Note that the paper recommends to project the local features using PCA. Just use Scikit-Learn's PCA for this purpose.

Write a code my-vlad that build the VLAD feature vector. For distance computations (see Sec. 3.1 in [b]), you can use built-in approximate distance computation functions from Scikit-Learn's (see eg [http://scikit-learn.org/stable/auto_examples/neighbors/plot_approximate_nearest_neighbors_scalability.html]. For dimensionality reduction (see Sec. 3.2 in [b]), you can use Scikit-Learn's PCA for this purpose.

You can now use my-vlad in place of get-bof in the previous homework, and run similar experiments for your full pipeline that is: 1) extract local features; 2) normalize local features; 3) learn vocabulary using my-kmeans; 4) build VLAD feature vectors for each example (signal/image); 5) learn Linear SVM; 6) predict on test set and evaluate performance.

To learn your vocabulary, extract the largest number of local features you can afford (time-wise). Reason backwards: if I take N local features, how much time my-kmeans will take to run? how much disk space can I

afford? Save the local features in your disk. Then, to choose the number of clusters, run experiments with the largest number of clusters you can afford. The larger the number of clusters, the better the feature representation. However, the larger the number of clusters, the slower the training of Linear SVM. Again, reason backwards. In a few words: be smart ;-)

Remark: for audio retrieval projects, replace the "Linear SVM" of your pipeline by approximate nearest-neighbors.

Best, --Z