CMSC 25025 / STAT 37601

Machine Learning and Large Scale Data Analysis

LSD PROJECT 1

Due: Monday, April 22 & Thursday, May 2, 2013

This version includes a more detailed specification of the second half of the project.

This is the first of four large scale data projects. In this project, you will work with the "Tiny Images" dataset. This dataset is made up of about 80 million small, 32×32 pixel images that were collected from Google image searches and query logs. See `http://groups.csail.mit.edu/vision/TinyImages/` for more on this data set.

This project explores several themes. Finding a good representation of data items is important. For images, working with the raw pixel values is typically useless. We need a representation that is better suited to comparing images in a meaningful way. In this project we use something called image "gists." Image data are also natural candidates for semi-supervised learning. In semi-supervised learning, we have a small number of labeled images, but a large unlabeled set of images. The hope is that these two types of data can be combined to learn more accurate classifiers.

To get a sense of some of the interesting possibilities with these data, you may want to peruse this paper: `http://cs.nyu.edu/~fergus/papers/fwt_ssl.pdf`. You are not required to read and understand this paper in any detail; it's pretty complicated. But what you will be doing in this project is very much in the spirit of this paper, and you will use the same dataset.

You can work individually, or in a team of two. We encourage you to pair up in a team. The project will be more fun that way, a bit less work, and you will learn more. See the course syllabus for more on working in a team.

The project will have two deadlines. The first deadline, for Problems 1, 2 and 3, is Monday, April 22 at 1:30 pm. Part of the reason is that your solution to Problem 1 will be used by your classmates in the second half of the project, which will have a deadline of Thursday, May 2 at midnight.

1. *Create an Evaluation Collection* (20 points)

   In this problem you will create an annotated subset of the tinyimages DAL. The goal is to get some cleanly annotated data that can be used for later problems. *Code for this problem is due on 1:30 p.m. on Monday, April 22.*

   You will find a notebook `lsd_project1_prob1.ipynb` in your home directory. This bit of code shows how to extract 2,000 tiny images of cats, and 2,000 images of maps. As you can see from scanning them, the images are not all cats and maps. Many of them are, but it's a jumble of stuff.

   The code lists 20 images that have been hand-selected as true cats, and 20 that are true maps.

Your task in this problem is to write a similar block of code, but for your choice of nouns. For example, you might choose "sunset vs. mountain" or "bee vs. flower." Construct a list of 2,000 images from each search. (If you don't get 2,000 results back, you made a poor choice of noun—try another.) Then, go through the images manually to select 100 of each category. For example, 100 pictures of bees and 100 pictures of flowers.

We will use these images to evaluate your classifiers in the later problems. Pass in your code as `lsd_project1_prob1.ipynb` in your `submissions/lsdproj1` directory.

2. *Compute Gists* (10 points)

Raw pixels are not a good representation.[1] We need a different representation if we have any hope of classifying images with reasonable accuracy. One well-known representation is due to Oliva and Torralba[2] and is called the "gist descriptor." We won't need to be concerned with the details. And fortunately, there is a Python module to compute this representation. Here is a bit of code that computes the gist descriptor of a tiny image:

```
def gist(index):
    import DAL
    tinyimages = DAL.create('tinyimages')
    img = scipy.misc.toimage( \
        tinyimages.byid(index).reshape(32,32,3, order="F").copy())
    return leargist.color_gist(img)

#calculate our gists
gists = dview.map_sync(gist, ids)
dview['gists'] = gists
```

This takes each tinyimage id, transforms the corresponding image to a standard jpeg format, and then calls a function from the pyleargist module that computes the gist descriptor. Each gist is a 960-dimensional vector.

Now, standardize the gists, so that each gist component (all 960 of them) has mean zero and variance one. The result will be a standardized gist representation for each of your 4,000 images.

3. *Calculate Pairwise Distances* (10 points)

Next, calculate the Euclidean distance between each pair of images in your collection of 4,000 images. To check the results, select a few of your true "cats" and your true "maps" (whatever your nouns may be), and compute the nearest neighbors—the images that are closest in Euclidean distance in "gist space." Describe the results. What does this tell you about the image representation? Would you get the same kind of results using raw pixels?

[1] `http://xkcd.com/722/`

[2] `http://people.csail.mit.edu/torralba/code/spatialenvelope/`

(Do you care to try it?) Include a discussion of these points in your iPython notebook, using markdown.

Pass in your code to both Problems 2 and 3 in the single iPython notebook `lsd_project1_prob3.ipynb` in your `submissions/lsdproj1` directory.

You will now construct kNN and semi-supervised classifiers on these data.

4. *Build a kNN classifier* (20 points)

   A k-nearest neighbor classifier is pretty much what the name suggests. For a given input x, you compute the k closest labeled examples to x. The predicted class of x is then the majority class among the k-nearest neighbors. To avoid ties, k is typically chosen to be odd. This is an example of a *nonparametric* method—it doesn't assume any simple parametric model of the data. The method is *inductive* because it works for any test point.

   In this problem you will construct k-nearest neighbor classifiers, using Euclidean distances computed with the gist descriptors that you extracted in the first part of the project.

   Using training sets of size $l = 10, 20, 30, 40, 50$, with half from one category and half from the other, compute the error rate of the kNN classifier on 100 of the remaining labeled images. Try several values of (odd) k.

   So, for example, with $l = 20$ labeled images of cats and maps, you choose a random subset of 10 labeled "true" cats and 10 labeled "true" maps. For each of the remaining 90 true cats (maps), you select a random subset of 50. For each of these cats (maps), you find the k nearest neighbors among the $l = 20$ labeled images, and select the majority label. This gives you an error rate on these 100 test examples, the percent predicted incorrectly. For each l, repeat this procedure multiple times to get an average error rate, together with the standard deviation.

   Give a plot of the error as a function of l, for each value of k that you use. How does the error rate vary with k? Do the same experiments on different pairs of "cats" and "maps" selected by your classmates. How do the error rates compare for different classification problems?

   Please note that each classification task is binary—you are discriminating cats from maps, or, say, bananas from starfish. You are *not* discriminating cats from "not cats."

   Place your code in a notebook called `lsd_project1_prob4.ipynb` in your directory `submissions/lsdproj1`.

5. *Construct a kNN graph* (20 points)

   To investigate a semi-supervised method, you will now need to construct a kNN graph. This is a graph with each node (image) connected to its k closest neighbors. But now you will construct this over all 4,000 of the images in your collection. Your goal is to chose k so that the graph is nearly connected. Specifically, chose k to be the smallest value so that at least

95% of the nodes are connected into a single component. (For the remaining $< 5\%$, you can connect them to their nearest neighbor in the connected component to get a connected graph.)

Construct the graph on some of the different subsets created by your classmates. How do the different values of k selected differ between the sub-collections?

6. *Compute the SSL harmonic function* (20 points)

   Now that you have constructed the kNN graph, you can experiment with semi-supervised learning using the harmonic function approach. Recall that this is an *transductive* method. For this part, you should follow the procedure for the kNN classifier in part 1 above.

   Specifically, for each $l = 10, 20, 30, 40, 50$, take a random subset of l labeled points, with $l/2$ true "cats" and $l/2$ true "maps." Now solve for the harmonic function over the remaining $4{,}000 - l$ nodes. Select a random subset of 50 of the remaining true "cats" ("maps") and select the label according to the value of the harmonic function on that node. This gives you an error rate on 100 random labeled examples that are not in your training set. For each l repeat this several times to get an average error rate, together with the standard deviation.

   How do the error rates compare to those obtained using the kNN classifier? Does semi-supervised learning give any benefit here? Plot the average error as a function of l. Do the same evaluation for some of the different subsets created by your classmates. Discuss your findings.

   Place your code for parts 5 and 6 in a notebook called `lsd_project1_prob6.ipynb` in your directory `submissions/lsdproj1`.