Memora: A Lifelog Search Engine

Blake Elias, MIT

Abstract—Lifelogging is a form of data collection where a peron wears a computer to capture data of their entire life, typically through photo or video data and possibly through physiological data. In this project I present Memora, a software tool for making sense of the data that one captures from their entire life history.

Index Terms—lifelog, scene recognition, classification, search engine

1 Introduction

For the past 3 months, I have been using a wearable device called the Narrative Clip (http://getnarrative.com) to capture my lifelogging data. The device contains a camera which is programmed to automatically take a photograph every 30 seconds, as well as a GPS location sensor which records its location along with every photograph.

This device and others like it promise to give people a "photographic memory" ability through which they will be able to relive and reflect on their life experiences. It seems plausible to expect this, since every moment that you experience will be captured and stored on a computer. Even if one photo every 30 seconds is not actually your *entire* life experience, looking back at the stream of photos from a day of your life 5 years ago would give you a very good idea of what happened that day - much better than what details you'd be able to recall without the photos. Using a lifelogging device could enhance memory by jogging memories which still exist in your mind but would otherwise never get activated. It could help you regain access to memories which, while you may have "forgotten about" them, you haven't actually "forgotten".

However, the device makers have hardly made the experience of viewing this data feel like having a photographic memory. As the size of the total collection increases, the feeling becomes more and more like having a barely-navigable list of files on a computer that you'd have to sift through endlessly in order to find the thing you're interested in. The time it would take you to find the precise moment you care about from a year's worth of photographs (about half a million of them), depending how well you're able to remember the specific date and time to look up, would likely make the experience overwhelming. If the task is too hard for one to even want to try, the experience of having a lifelog would be unexciting.

On the other hand, within this data set lie many special memories from our lives, exciting experiences worth sharing with friends and future generations, and possible insights into how we can live better. If we hope to turn a pile of photos into a machine-enhanced superpower of photographic memory, then we need software tools for browsing these photographs in a way that is more intuitive to humans, rather than computers. The computer's easiest querying scheme is based on the name of the file, the day it was taken on, and other relatively generic, unemotional features. The human's easiest querying scheme, which we use for our own memory, is a lot more personal: where we were, who we were with, what we were doing, and so on. When you and a close friend want to talk about an experience you shared some time in the past, you bring up the memory in your friend's mind by telling a story with a bunch of features, possibly in chronological sequence. As a simple example, "remember the time when we were with Sally in her apartment before going to the Red Sox game?" The listener almost instantly knows what you are talking about and has some image, phrase, or other experience now re-

playing in their head. They are able to perform this task because their mind has stored the memory in a way that is retrievable with the type of language you've used to describe it. Computer programs don't yet store memories in the same way that we do, but in this project I demonstrate a way that an algorithm can begin to match our intuitive structure of memory. Using Memora, I can ask the computer "show me the first lecture of my 6.867 class", and with some supervised training, have it understand which moments match my query.

2 Feature Selection

I assign features to photographs based on whose faces are present in a photo. Each point (a photo is one point) is assigned a bit-vector of people who are present in the photograph. If there is a total of N people whose faces appear in my dataset, each photograph's feature vector is a length-N bit vector where a person's slot contains a 1 if their face is recognized in the photo, and a 0 if not. I was able to recognize faces using Facebook's face recognition algorithm as well as the one in scikit-learn, however I did not use the results from these algorithms in my final implementation due to some implementation challenges. I hand-labeled faces in the photographs I wanted to classify, realizing that with more time spent on the engineering of this system the faces could just as well have been labeled by a machine.

3 Activity Classification

The first problem that I solved is training an algorithm to recognize common scenes that I experience regularly. For example, the algorithm should be able to differentiate between times I've been in one class versus another, versus when I'm at home with my family.

I have solved this using supervised learning: I select scenes from the timeline and mark them with particular labels - eg. a 1.5-hour block for a 6.867 lecture, a 1-hour block for another class, and a 15-minute block at home with my friends. Since each 30-second interval is one point in the original set, I combine features from longer blocks by taking the union of the set of faces recognized in each photo (for this classifier, I used faces as the only features in the classifier), so that a long scene has a feature set that looks like any normal point. Then, I run a nearest-neighbors classifier to classify every photograph in the timeline. I used scikit-learn's built-in k-Nearest-Neighbors algorithm with $k = 1$ (each point gets the same label as its nearest neighbor), with Jaccard distance as the distance metric (number of dimensions where the values are non-equal in the two bit-vectors, divided by the number of dimensions that are non-zero in at least one of the two vectors).

This algorithm worked very well, but with a limited scope. 1-NN with Jaccard distance correctly classifies all the photographs which contain at least one face that is also found in one of the labeled scenes. Photos that have only new faces (people who are not in any labeled scene) get assigned to an arbitrary class, which is wrong. Furthermore, nearest-neighbors cannot classify photographs that have no faces in them, since the bit-vector is all 0s - I had my algorithm skip these, but if they were classified they would also end up in an arbitrary, wrong class (I'll come back to this in the "Activity Transitions" section). But let's ignore these problems for a moment. Even the ability to classify photos that do have faces in them, while it worked well in my small example, would not necessarily be robust to all situations - for example, there was no person in my data set who showed up in two scenes that had different labels. Had there been any overlap of this sort, the algorithm would likely start making mistakes. It could see a photo with one friend of mine and classify it as either a class we have together or dinner with a group of friends who we often eat with. Solving this would likely require using a higher value of k - that is, to use more neighbors in the classification.

4 Activity Transitions

Next, I tried using a Gaussian HMM to infer a scene based on the faces in each photo, without training on any labeled data. Here, the hidden variable is the scene label, and the observation is a bit-vector of all the faces that were recognized in each photo. I filtered for photographs

that had faces in them, and then passed these as observations to the HMM. Since the HMM is unsupervised, it does not know any examples of which photos belong to which category - the only information it was given was the number of different values that the hidden state should be allowed to take on. I took a subset of the images I had labeled, counted the number of different labels I had given to the photos, and then added 1 to allow for the possibility of a state that represents all the 'in-between' photos - photos that don't belong to any of the scene classes but instead lie in transitions from one activity to the next - things like walking in a hallway, riding a train and so on. In the subset of images I chose, there had been three labeled activities so I set the number of states equal to 4. I estimated an initial value of the transition matrix as follows:

$$\begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.7 \end{pmatrix}$$

These initial values were appropriate in order to make the hidden state usually stay where it is (because I don't typically switch back and forth between scenes many times - from minute to minute, I usually stay in the same activity like a class, and then once every hour or two I will move to a new class or activity).

I fed the observations and my initial guess of the transition matrix into the Gaussian HMM provided by scikit-learn, and used its built-in "fit" procedure to estimate the model parameters and hidden states using EM. After running it 1000 times, I selected the model with the highest score and found a sequence of hidden states which only misclassified 4 photos out of 40 from my original scene labeling (that is to say, photos which I had marked as belonging to one class (though the HMM didn't know that), and which the HMM put in another class). Though it isn't defined which hidden state in the HMM corresponds to which label in my original labeling, it is pretty easy to tell by inspection when looking at the results, as photos with the same original label mostly received the same hidden state from the HMM, and each different label got a different hidden state as the most common state among its photos.

What did not work so well was the states assigned to the images I had not labeled. These photos have miscellaneous faces in them who were not relevant to any of the activities that I had labeled. Only 3 of the non-labeled images were assigned to the "other" state, out of 23 - the rest were assigned a hidden state that was the same as the state for one of the labeled scene categories.

5 Results Illustration

Manually labeled classes:

```
[A A A A A A A A A A - - - -
 B B B B B B B B B B - - -
 - - - - - - - - - - - - - -
 - - C C C C C C C C C C C
 C C C C C C C]
```

Hidden state assignments in the HMM, unsupervised:

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1
 3 3 3 3 3 3 3 3 3 3 3 3 0 0
 0 1 1 1 1 1 3 3 3 3 3 3 3 3
 3 3 1 2 2 2 2 2 2 1 1 2 2 2
 2 2 2 2 2 1 2]
```

6 Contributions

(1) A scene-classification model that classifies scenes using labeled examples, using face recognition to generate the features.

(2) Another scene-recognition approach which can group images together into scenes with no labeled training examples at all. Unlike other common scene-recognition approaches, this approach takes advantage of the similarity between images that were taken at around the same time, rather than just using the features of one image.

7 Discussion

While these models don't fully solve the problem of recognizing activities from daily life, they show some promising results that such a task is feasible to be done by computers with relatively straightforward application of machine learning techniques.

The nearest-neighbors classifier could be useful when one is browsing lifelog photos and wants to see all scenes of a particular category, where they have told the machine about some other scenes of that category before. The HMM approach could be used for gathering photos into scenes without the machine necessarily knowing what the scenes are. This would be useful in a photo-browsing interface to facilitate tagging of scenes, since most of the images belonging to a scene could already be grouped together, rather than the user having the tedious task of locating boundaries between scenes. The HMM could also be useful in a photo-browsing interface to reduce redundancy of similar photos. For example, if one wants to play through an entire day's worth of photos (1000-2000 photos), a browsing interface could animate the photos as frames at a high speed, increasing the speed when a range of consecutive images all have the same hidden state, and slowing down the speed during transitions from one state to another.

8 Future Work

I plan to try the following possibilities to improve the accuracy of both of these models. First is adding more features from the data that I have. GPS coordinates are available from the Narrative Clip at every moment when a photo was taken. This would help a great deal in differentiating photos from different scenes, since moving from one location to another one is highly likely to represent a transition from one activity to another (extraneous movement might have to be removed for things like going to the bathroom in the middle of a class, if the end-user does not want this classified as a separate activity), or as an activity in itself (such as running, biking, or moving due to transportation). (The only reason I did not use GPS in this paper was that the Clip doesn't fully compute the coordinates, it just records some measurements to save battery on the device, and leaves it up to the developer to map these measurements to coordinates - I opted not to do investigate this in the interest of time, but it certainly would help.) I would also consider adding other image features, such as values from the color histogram - I believe that this would readily detect changes in environment and help group images together that belong to the same scene. This would reduce some of the extraneous transitions that the HMM generated when faces would come and go, since the histogram would remain largely unchanged due to the constant background in the images. Other features worth adding would be object labels using an object-recognition library - this would allow for classification based on objects you've seen, like furniture, plates and silverware, and so on.

Another future technique to try is to smooth out the appearance of features using a moving window. Specifically, if a person's face shows up in a photo, then is absent in the next five photos but is present again in the next photo, my current model encodes that sequence as [... 1 0 0 0 0 0 1 ...], which makes no indication that the person was present in the five middle photos. Instead, this could be encoded as something like [... 1 0.7 0.5 0.4 0.5 0.7 1], which would provide some indication that the person was present in those in-between moments. This would help remove some of the extraneous transitions of the HMM, and could even help with the kNN classifier since it would be able to synthesize the concept of a group of people being together during some time range even if no one photo contains all of those people at the same time.