Project 1: Detecting Pointing Gestures in Video

CS 7626/4403 Introduction to Behavioral Imaging February 14, 2018
Spring 2018
Instructor: Dr. James M. Rehg
Lead Teaching Assistant: Ahmad Humayun

Background

A key aspect of early child development is the acquisition of nonverbal communication skills. Long before they can talk, children coordinate their gaze, facial expressions, and body movements to communicate their intentions, feelings, interests, and requests to their caregivers and peers. Gestures are a key aspect of nonverbal communication. By pointing to an object, for example, a child can communicate their interest in it to a social partner, thereby creating a shared reference. As illustrated in Figure 1(a), a child might point to a toy to communicate their excitement or enjoyment. This is an example of joint attention, a triadic interaction between a child, a social partner, and an object such as a toy [2]. We know that joint attention is an early milestone for social communication, and that it's impaired in children with autism. In fact, children's responsiveness to bids for joint attention from others has been shown to predict their rate of language growth [5].

Figure 1: Communicative gestures: (left) Pointing at a toy; (center) Showing a toy; (right) Requesting a toy. All images taken from [3].

Procedures have been developed for manually coding communicative gestures from video [3], and video coding by multiple human raters has provided the data to support findings about early communication in autism. However, it would be extremely useful if automated measures for nonverbal communication could be developed, as this would enable behavioral assessment on a much larger scale than is currently possible.

Recent progress in computer vision on the detection of body poses from images, via methods such as OpenPose [1], have created the potential to automatically identify communicative gestures. However, this is not a simple task, as gestures with very different meanings, such as

requesting and showing, can produce similar limb configurations. See Figure 1 for some examples.

Goal

In this project, you will *develop a detector for pointing gestures* by analyzing the output of OpenPose [1]. You will be using videos collected at Georgia Tech as part of the Multimodal Dyadic Behavior Dataset (MMDB) [4]. Each video depicts a short (2-3 minute) interaction between a child and an examiner. All of the pointing gestures produced by the child have been manually annotated, with starting and ending times recorded for each gesture. We will provide you with these annotations, along with the OpenPose output for each of 33 videos. Most video frames contain multiple body poses, because the child is frequently sitting on their parent's lap and the examiner is often visible. We have identified the child body pose, whenever it exists, in each video frame and will provide you with that information. We will also enable you to watch the MMDB videos with the OpenPose skeletons superimposed over each video frame, so you can identify and analyze any errors in the OpenPose output. We will provide further instructions on how to access these videos in a subsequent announcement.

Important Note: *None of the resources for this project, including all of the files that we give you, can be distributed to others or posted online in any form. Doing so will be considered an Honor Code violation*. ***These resources are for your use in this project only, and you must delete all data you receive from us at the end of the semester***. If you wish to access any of this data for research purposes beyond the semester or outside this class, we welcome your interest, and we will make arrangements to support you under a new DUA. If you have colleagues who would like to access this data, please put them in touch with us. Likewise, we are not giving you access to the original MMDB videos for this project. If you want access to these videos we can provide it, but it will require a different arrangement. If you have any questions about what is allowed and what is not, don't hesitate to post on Piazza or contact the instructor or TAs.

Procedures and Deliverables

The project has five parts, and each part has a deliverable associated with it. The deliverables are slides, and you must submit a single PDF file containing all the slides for your project on T-Square, along with your source code. See below for the submission instructions.

Preliminaries

For each of these parts, you will be developing a classifier that takes the OpenPose keypoints and makes predictions about whether or not the child is performing a pointing gesture. Thus the core of the project is developing a binary classifier. You are free to use any modern classifier design for the classification task. Some valid choices are support vector machine (SVM), random forest, and boosted decision trees. *Choices that are not allowed are simple, generally-inferior methods like nearest-neighbor and logistic regression*. You are free to use any software

package or implementation of these methods. Scikit-learn is a good choice for Python, the Weka library is another option. If you have any doubts or questions about the classifier please post on Piazza.

Note that your classifier will have a threshold that trades off between the two types of errors (false positives and false negatives). For each of the five parts below, you will perform cross-validation to assess the accuracy of your classifier. For each cross-validation split, you must compute the precision-recall (PR) curve for the resulting classifier. The Average Precision (AP) provides a single number summarizing the PR performance. You will compute an overall performance number for your classifier by taking the average of the APs from each split. *We will tell you which splits of the data to use for your final project submission.* We will review these topics briefly in class.

Part 1: Single-frame Arm-based Detector

The first part of the project involves training a detector that classifies each frame independently as to whether the child is pointing or not using just the keypoints corresponding to the child's arm. This is the simplest pointing detector, as it does not try to use information about the pose of other parts of the body besides the arm, and it does not integrate multiple frames over time.

Your detector should output a single label (pointing or not-pointing) given the pose data for each frame using your trained classifier. In making this determination, you will obviously need to apply your trained detector to both arms. Each frame also has a ground truth label of positive or negative based on the annotations (i.e. all frames from the start to the end of a pointing annotation are positive). For each cross validation split, you will compute the frame-level AP from the PR curve. You will report your classifier choice, classifier implementation, and average AP, along with your software implementation, as your deliverable for part 1. Note that when keypoints are missing (i.e. when OpenPose is not confident about a keypoint), OpenPose will output zeros for their coordinate locations. It's important that the feature vector always be the same length. Your classifier should be able to learn to ignore the zeros when they are present.

Part 2: Single-frame Body-based Detector

The second part is identical to the first part, except that you will use the entire vector of keypoints from the child's body to detect the pointing gesture, rather than just the arm by itself. All other aspects are the same.

Part 3: Sliding window Arm-based Detector

For this part, you will return to the arm-based feature model of Part 1, but now your classifier will be designed to work over a sliding window. Your implementation should be parameterized so the window can be of variable length and variable stride. *Window length* is the number of consecutive frames that are combined to perform classification. *Window stride* is the number of frames separating the start of each window position. So with a length of 5 and a stride of 2, the

first window position would process frames (1, 2, 3, 4, 5) and the second window position would process frames (3, 4, 5, 6, 7). In order to compute the accuracy of your method, you will need to assign each window position a ground truth label of either positive or negative. This will be done by computing the overlap between the sliding window position and the annotated pointing segments. Your code should have a variable, *overlap percentage,* that specifies how much overlap with the ground truth is needed for a sliding window to have a positive label. For example, with an overlap of 50%, a window is labeled positive if 50% or more of its frames are labeled as pointing according to the ground truth annotations.

You will be averaging APs across splits to compute a single performance measure as you did before. However, in this case the classifier decisions are being made at the window level rather than at the frame level. The recommended way to generate a feature vector for each sliding window position is by simply stacking the keypoints from multiple frames together into a single vector.

Part 4: Sliding window Body-based Detector

This part is just like Part 3, except that you will use the entire vector of keypoints from the child's body to detect the pointing gesture, rather than just the arm by itself. All other aspects are the same.

Part 5: Final Detector

For this part, you need to try *at least one additional idea*, beyond what you have already done in parts 1-4, to improve the detector. For evaluation purposes, we will provide you with code to compute the AP of the detector in a way that penalizes multiple overlapping detections. This code would provide a more accurate measure of the localization performance of a detector, and is routinely used for detection problems in computer vision. You will apply the AP function to your best-performing method from Parts 1-4, and that will be the baseline you compare against in Part 5. Your task is to beat your baseline if possible! Here are some suggestions for things you might want to try:

- Consider how the errors in the OpenPose output affect your classification performance. Is there some way to utilize the confidence scores that OpenPose provides for each keypoint to improve the classifier? Is there some preprocessing or filtering that you could do to reduce the impact of OpenPose errors? For example, could temporal smoothing or methods for filling in missing keypoints help?
- Look at the temporal evolution of the pointing gesture. Could adding features that encode the velocity of the keypoints improve the performance? Experiment with adding additional temporal features that capture the motion of the arm and hand.
- We will provide you with annotations for the requesting and showing gestures in the sessions that you are analyzing. These gestures can be easily confused with pointing. You can experiment with training a multi-class classifier to reduce the confusion.

- Consider the impact of the window length and stride on the performance. Could you generate window positions that are flexible and are based on the data rather than always being a fixed size? Such adaptive windows are called proposals. One way to generate proposals for temporal data is through change-point detection. Given windows of a variable size, Fisher vector encoding can provide a feature representation.

Deliverables

Each team leader will submit the deliverables on behalf of their project team by the deadline. Each member of a project team will receive the same grade. There are two deliverables:

- A PDF document containing slides for each part. *We provide a Powerpoint template which you should use to create your deliverable slides. It's important to* ***use our provided template*** *because there are several slides in which you must answer some questions (as well as provide information about your team). Fill in the template and save it as PDF.*
- The source code that you wrote for each part

Starter Code and MMDB Pose Data

We are providing you with OpenPose [1] results on 33 different sessions from MMDB [4]. You would have access to the child's pose in every frame of the video, if available. A pose is a list of 2D coordinates for pre-defined key-points on the body. OpenPose data includes 18 body pose key-points, such as the shoulder, elbow, neck, etc. Additionally, OpenPose also provides fine grained predictions for 21 key-points on each hand. You can find more details about these key-points here. We also provide a scalar value indicating how confident OpenPose is in its prediction for each of the 60 key-points. You can use the following script to visualize the OpenPose data for the child in a selected session.

You will download three files from the T-Square assignment page: `openpose_results.zip`, `sess_vid_meta.npy`, `starter_rabc.py`. Download them to a single folder. Unzip openpose_results.zip in the same folder, so that your folder structure looks like:

- `sess_vid_meta.npy`
- `starter_rabc.py`
- `openpose_results/`
  - `RA025_complete`
  - `RA030_complete`
  - `...`

The `openpose_results/` directory stores the open pose results for all 33 sessions. Each directory name inside `openpose_results/` indicates a session identifier (like `RA025_complete`).

The starter code is written in python. To run it you would need to install opencv and numpy for python. The code has been tested on python 2.7.12 (it has not been tested on python 3). Once python with opencv/numpy is installed, you can view OpenPose results by providing the session identifier as an argument:

```
>> python starter_rabc.py -v RA025_complete
```

This will open up a window displaying the pose of the child in the first frame of the session. You can iterate over the frames in the session using the keys printed on the console. To ease navigation, the frame number is printed on the top left corner. To view just the key-points for the arms and hands, use the -a flag:

```
>> python starter_rabc.py -v RA025_complete -a
```

Whenever the script shows a frame where the child is "pointing," the window would turn gray, and the info area would say "+ pointing." You can switch off this behavior by:

```
>> python starter_rabc.py -v RA025_complete -n
```

References

[1] Cao, Z., Simon, T., Wei, S.-E., & Sheikh, Y. (2017). Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). http://doi.org/10.1109/CVPR.2017.143

[2] Carpenter, M., Nagell, K., Tomasello, M., Butterworth, G., & Moore, C. (1998). Social Cognition, Joint Attention, and Communicative Competence from 9 to 15 Months of Age. *Monographs of the Society for Research in Child Development*, 63(4), 1–174.

[3] Mundy, P., Delgado, C., Block, J., Venezia, M., Hogan, A., & Seibert, J. (2003). A manual for the abridged Early Social Communication Scales (ESCS).

[4] Rehg, J. M., Abowd, G. D., Rozga, A., Romero, M., Clements, M. A., Sclaroff, S., (10 others), Ye, Z. (2013). Decoding Children's Social Behavior. Proc. of *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3414–3421. http://doi.org/10.1109/CVPR.2013.438 MMDB website:

[5] Siller, M., & Sigman, M. (2008). Modeling longitudinal change in the language abilities of children with autism: Parent behaviors and child characteristics as predictors of change. *Developmental Psychology*, 44(6), 1691–1704. http://doi.org/10.1037/a0013771