CS228: Human Computer Interaction

Deliverable 6

Description

In this deliverable, you will expand your k Nearest Neighbor classifier such that it takes as input training data for all of the ASL numbers between zero and nine, from all of your fellow classmates. You will save your classifier to a file, and load it in to your program that simply draws the user's gestures. For each frame of captured data, the classifier will now output a number in $[0, 9]$, which indicates which of the 10 ASL numbers the classifier thinks the user is currently signing. A demonstration of a successful implementation of this deliverable is here. (Ignore the 3D visualization.)

Instructions

1. Create a new directory called `Del6` within your CS228 directory. We are creating a new directory here so you can easily revert to earlier deliverables.

2. Create a subdirectory in it called `userData`.

3. Copy your four data sets into `userData`.

4. Copy `Classify.py` from the previous deliverable into `Del6`. Make sure that it is added to your repository.

   Note: you are gradually going to be adding files to this directory, which sits within the CS228 directory, which in turn, for most of you, sits within Leap's `lib` directory. So, throughout this deliverable, you may need to alter your programs in `Del6` to tell them where to find the Leap libraries. This may require you to change the

   ```
   sys.path.insert(0, '..')
   ```

   line to

   ```
   sys.path.insert(0, '../..')
   ```

   to reach not the parent directory (`'..'`) but the grandparent directory (`'../..'`).

5. Add the following line at the bottom of `Classify.py`, which will save your kNN classifier after it has been created:

   (a) `pickle.dump(knn, open('userData/classifier.p','wb'))`

6. Run `Classify.py` and make sure that this file has been saved in `userData`.

7. Copy and paste your Python program from Deliverable 2 into `Del6`. Add it to your git repository. (To run, this program calls other programs you have written. Copy them into `Del6` as well and add them to your repo.) Recall that this program just draws each frame of Leap Motion data to the screen. For simplicity let's call this program `Del2.py`.

8. We will now modify this file to (1) read in your classifier; (2) capture the 30 feature values that this classifier is expecting from each frame of data; and (3) make a prediction about which of your two digits is being signed by the user during each frame. To start with this restructuring, load in your classifier at the top of `Del2.py`:

   (a) `clf = pickle.load( open('userData/classifier.p','rb') )`

9. Run `Del2.py` and make sure that it still works correctly.

10. Next, create a 1×30 numpy vector to store the 30 feature values from each frame of Leap Motion data. Do this by placing this line immediately after line 8a:

    (a) `testData = np.zeros((1,30),dtype='f')`

11. Next, we need to fill this vector whenever a new frame of data is captured from the Leap Motion device. Add lines (c) and lines (h)-(m) below to `Del2.py`:

```
(a) frame = controller.frame()
(b) if ( len(frame.hands)>0 ):
(c)       k = 0
(d)       for finger in ...:
(e)           ...
(f)           for b in ...:
(g)               ...
(h)               if ( (b==0) or (b==3) ):
(i)                   testData[0,k] = xTip
(j)                   testData[0,k+1] = yTip
(k)                   testData[0,k+2] = zTip
(l)                   k = k + 3
(m)       print(testData)
```

    Note that you will either have to define `testData` as a global variable, or pass it as an argument through several functions. Either way is fine.

    Note also how the above code gradually fills in just the tip positions (lines (i)-(k)) from only the first and fourth bones (line (h)) to the `testData` vector. Each time a new frame of data is captured, the vector begins to be filled in from its first element again (line (c)).

12. Run `Del2.py`. Whenever your hand is above the device, you should see a vector with 30 numbers that is fully filled (i.e. there are no zeros).

13. Immediately after line 11m, include this line

    (a) `testData = CenterData(testData)`

    Now write a function with this name. This function should pull out each x coordinate in the vector, compute the mean value across these x coordinates, and subtract this mean from each x coordinate in the vector. It should then perform the same operation for the y coordinates and the z coordinates.

    Hint: All of the x coordinates are stored at the first, fourth, seventh, etc. positions in the vector, which can be accessed with `testData[0,::3]`. All of the y coordinates are stored in the second, fifth, eighth, etc. positions in the vector, which can be accessed with `testData[0,1::3]`. The same applies for the z coordinates.

14. Now you are ready to collect predictions from your classifier. After line 13a, paste this

    (a) `predictedClass = clf.Predict(testData)`

    (b) `print(predictedClass)`

    (You can also delete line 11m.) When you run your program now, you should see integers continuously printed whenever your hand is over the device. Sign each of the two numbers assigned to you; does your classifier recognize them? What happens if you extend your fingers as far as they will go, compared to letting them droop? What happens if you change the orientation of your hand? What happens (if your digit involves multiple extended fingers) if you spread your fingers apart or allow them to lie next to one another? If the predictions are poor, go back through the steps above, or try re-recording data for these two numbers using the programs you developed in Deliverable 5.

15. Now, let's expand your classifier to recognize three, instead of two numbers. Download all of the digit data sets from here into `userData`.

16. Unzip this file inside of `userData`. Make sure that you now have a large number of .p files in this directory. (Depending on how you unzip this file, they may be placed in a sub-directory within `userData`; move the files out of there and into `userData`.)

17. Now we're going to do some work on your `Classify.py` file. Modify `Classify.py` (if you haven't already) so that it reads in your four data sets.

18. Read through the names of the .p files in `userData`. Choose **one** digit that is different from your two digits, and that is from another student. (Please choose a student from Table 1 who does not have an asterisk next to their name.) For example if you were assigned digits 0 and 4, you might choose `Clark_train6.p` and `Clark_test6.p`.

19. To read this new digit in, read the training data and testing data corresponding to that digit into a fifth and sixth matrix in `Classify.py`. For example,

(a) `train6 = pickle.load(open('userData/Clark_train6.p','rb'))`

(b) `test6 = pickle.load(open('userData/Clark_test6.p','rb'))`

(Note that you should make sure that your data matrices are named trainM, testM, trainN and testN if you were assigned digits M and N. For example if you were assigned digits 0 and 4, your data matrices should be named train0, test0, train4, and test4.)

20. Run your program and make sure that you get the same prediction accuracy you did before this step (you have not made use of these two additional matrices yet).

21. Reduce and center these two new matrices by sending each of them to `ReduceData(...)` and `CenterData(...)`. Run your program and make sure its behavior does not change.

22. Modify `ReshapeData(...)` to take three matrices rather than two. In your first call to this function, pass the three training data sets; in the second call to it, pass in the three testing sets. Run your program and make sure that it behaves exactly the same (we have not yet changed the behavior of `ReshapeData` itself).

23. Inside `ReshapeData(...)`, expand the X and y data structures to hold 3000 rather than 2000 samples. Copy data from the two new matrices into `X[2000:2999,:]` and `y[2000:2999]`. Make sure to set the elements in y to the new digit that you've included.

24. Run your program, and you should see the prediction accuracy change a bit: your classifier is now trained to recognize three of the first 10 ASL numbers.

25. Run `Del2.py`, and sign each of these three digits in turn: does your program correctly predict all three?

26. Repeat steps 18 through 25, this time including a fourth, different digit. This time through, you may want to modify your code so that only a few changes are required to add a new digit. This is because in the next step...

27. Repeat step 26 for the fifth digit, then the sixth digit, and so on, until you have included all 10 digits.

28. Record a video like this one, upload it to YouTube, and submit the resulting URL to BlackBoard. Your program does not have to be perfect, but it should have about the same accuracy as seen in the reference video.

    Optional: Machine learning algorithms usually do better when they are provided with more training data. So, if you want to, you can include multiples instances of the same digit, imported from different students. If you've structured your code well, it should require just a few modifications to read in *all* of the data. Good luck!

Table 1: CS228 2019 class data. If there are two asterisks next to your name, redo Deliverable 5. If there is one asterisk next to your name, try re-recording your data and re-training your kNN.

Index	Name	D1	D2	Before cutting	After cutting	After centering
1	Kevin Yeung	6	7	79.94%	98.75%	99.85%
2	Jordan Deso	5	6	61.10%	99.85%	98.75%
3	Sida Liu	2	3	90.75%	99.85%	98.75%
4	Gordan MacMaster	6	7	77.90%	87.60%	100.0%
5	Hengjun Wu	5	6	89.85%	92.85%	92.10%
6	Zachary Ward	3	4	78.35%	97.60%	100.0%
7	Sam Clark	0	1	89.50%	90.60%	92.25%
8	Luke Beatty	3	4	75.45%	97.05%	100.0%
9	Anna Erickson	7	8	72%	77%	87%
10	Rendong Zhang	7	8	82.9%	93.8%	97.4%
11	Josh Childs	9	0	75%	86%	92%
12	Sam Zonay	8	9	98%	99.4%	99.75%
13	Eve-Audrey Picard	6	7	71%	77%	100%
14	Breanna Apple	2	3	68.35%	89.70%	100.0%
15	Jeff Giroux	1	2	93.5%	100.0%	100.0%
16	Mary Livingston	4	5	70.6%	76.85%	94.85%
17	Rose Warren	4	5	71.9%	95.7%	100%
18	Krystal Maughan*	8	9	30%	50%	?
19	Josh Newton	1	2	94.6%	97.2%	100.0%
20	Will Soccorsi	9	0	87.2%	93.3%	98.25%
21	Eilliott Gear*	0	9	67%	73%	62%
22	Luke Trinity	2	3	90.8%	92.25%	96.9%
23	Eric Boland	5	6	99.95%	99.95%	99.95%
24	Connor Burleson	7	8	68.2%	89.0%	98.3%
25	Carlos Castrejon Sanchez*	8	9	74.75%	78.95%	79.2%
26	Jing Lu	5	6	71.8%	100.0%	98.3%
27	Anyi Huang	6	7	70.1%	74%	86.4%
28	Nathaniel Ogilvie	3	4	62.85%	82.35%	93.1%
29	Brandon Lee	9	0	73.65%	94.15%	100.0%
30	Jordan Genovese	0	1	98.3%	100.0%	100.0%
31	Filip Saulean	8	9	69.25%	79.45%	100.0%
32	Xuanbang Liu	3	4	64.65%	100.0%	100.0%
33	Shawn Beaulieu	4	5	94.5%	74.6%	81.8%
34	Eric Langdon	8	9	73.6%	82.1%	83.5%
35	Tripp Gordon	2	3	72.85%	73.0%	96.15%
36	Jonah Rubin	7	8	84.4%	91.75%	99.0%
37	Joseph Ortigara	4	5	85.0%	94.7%	97.5%
38	Christian Deluca	4	5	65.75%	65.65%	98.3%
39	Stover Mardis	7	8	83.25%	100%	100%
40	Will Fath	8	9	66.2%	67.5%	100%
41	ZhaoJun Lin	0	1	60.9%	70.2%	93%
42	Max Peck	5	6	92.2%	98.95%	100%
43	Owen Bruning	6	7	85%	89%	94%
44	Dave Landay*	7	8	81.65%	38.3%	21.2%
45	JJ Thissell Jr.	1	2	80.7%	91.4%	99.75%
46	Codie Cottrell	1	2	75%	84.75%	89.35%