CS228: Human Computer Interaction

Deliverable 6

Table 1: Assigned numbers

Index	Name	D1	D2	Before cutting	After cutting	After centering
1	Monica Bilodeau	4	9	71.45%	81.80%	82.25%
2	Chaquille Browne	0	5	88.00%	89.32%	91.40%
3	Lily NguyenB4	7	8	70.40%	98.25%	97.20%
4	Bennet Siegel	3	5	95.25%	100.0%	100.0%
5	Marco White	5	7	85.45%	95.25%	99.55%
6	Hayden Lam	1	8	60.05%	99.20%	100.0%
7	Eric Clark	6	8	29.85%	38.75%	91.55%
8	Anna Margolis	0	1	96.30%	99.45%	99.40%
9	Grace Samsonow	1	3	51.35%	98.95%	99.30%
10	Gabe Lewis	0	9	77.55%	99.55%	99.70%
11	Jason Sheehan	2	4	98.65%	100.0%	100.0%
12	Kevin Mee	2	3	59.15%	86.95%	93.70%
13	Sean Nealon	5	6	93.30%	90.50%	85.70%
14	Dilan Kiley	6	3	78.55%	100.0%	100.0%
15	Sara Piette	8	9	71.55%	39.80%	73.85%
16	Brandt Newton	6	7	59.55%	61.20%	57.15%
17	Jackson Flore	3	6	67.20%	75.60%	100.0%
18	Dillion Belive	2	5	31.65%	100.0%	100.0%
19	John Montfor	4	5	00.00%	00.00%	00.00%
20	Drew Guild	5	4	73.05%	81.40%	81.40%
21	Henrique Ch	6	2	74.90%	100.0%	100.0%
22	Keith Miller	3	4	89.05%	93.08%	99.10%
23	Sandra Rome	0	2	00.00%	00.00%	00.00%
24	Daniel Grzen	4	5	75.70%	90.50%	93.40%
25	Matthew Bea	1	3	100.0%	100.0%	100.0%
26	Owen Marsh	1	2	82.60%	87.80%	93.95%

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.

Instructions

1. Create a new directory called `Del6`.
2. Create a subdirectory in it called `userData`.
3. If you saved your training and testing data sets in the previous deliverable using the `pickle` package, please skip to step #5. Otherwise, go to the next step.
4. In order to make sure that everyone's data is in the same format, please make sure that your data is saved using the `pickle` package. If you saved your data files in Deliverable 5 using `numpy.save`, there are two ways you can do this (either is fine):
   (a) Go back to Deliverable 5, and replace each call to `np.save` with `pickle.dump`, and each call to `np.load` with `pickle.load`. Then, re-record the training and testing data sets for your two digits.
   (b) Create a conversion program that takes as input `numpy` data structures and re-saves them using pickle:
      i. `import numpy as np`
      ii. `import pickle`
      iii. `data = np.load('userData/dataFile.dat')`
      iv. `pickle.dump( data , open('userData/dataFile.p','wb') )`
5. Copy your four data sets (in the `pickle` format) into `userData`. Make sure that these files have the `.p` extension to signal that they are in this format. (If you saved them using `pickle` with the `.dat` extension, just rename the files.)
6. Copy `Classify.py` from the previous deliverable into `Del6`.
7. Add the following line at the bottom of `Classify.py`, which will save your kNN classifier after it has been created:
   (a) `pickle.dump(clf, open('userData/classifier.p','wb')`
8. Run `Classify.py` and make sure that this file has been saved in `userData`.
9. Copy and paste your Python program from Deliverable 2 into `Del6`. Recall that this program just draws each frame of Leap Motion data to the screen in 3D. For simplicity let's call this program `Del2.py`.
10. 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') )`

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

12. 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 10(a):

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

13. 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 i in range(0,5):
(e)           ...
(f)           for j in range(0,4):
(g)               ...
(h)               if ( (j==0) | (j==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 how this 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)).

14. 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).

15. Immediately after line 13(m), 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]`. You get the picture for the z coordinates.

16. Now you are ready to collect predictions from your classifier. After line 15(a), paste this

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

    (b) `print predictedClass`

    (You can also delete line 13(m).) 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.

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

18. 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`.)

19. 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 using the `pickle` package.

20. 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. For example if you were assigned digits 0 and 4, you might choose `Clark_train6.p` and `Clark_test6.p`.

21. 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.)

22. 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).

23. 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.

24. 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).

25. 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.

26. 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.

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

28. Repeat steps #19 through #27, this time including a fourth digit.

29. Repeat step #28 for the fifth digit, then the sixth digit, and so on, until you have included all 10 digits.

30. Record a video like this one, upload it to YouTube, and submit the resulting URL to BlackBoard.