6.835: Mini Project 2

Nada Hussein

Spring 2020

1 Nearest Neighbor

Question 1. Look at several sequences of each type of gesture using the GUI. What are some of the differences in how gestures of the same type are performed? Note that the poses have already been normalized to have the same shoulder width and rotated so that the average direction is facing directly at the camera. What difficulties would you encounter if this were not the case? Describe the difficulties specifically faced by a Nearest Neighbor classifier.

In gestures of the same time, I could see that there were variations in the speed that the motion was performed, as well as changes in how the hands moved relative to each other and how the joints were angled. Some sequences performed the gesture very quickly while others took a while before they began moving, or just did the gesture as a whole slower. Furthermore, sometimes the amount that a user tilted their hands (i.e. in the rotate gestures) varied in that some others had a much more extreme rotation of their hands and arms than others.

Because nearest neighbor uses a standardized distance metric, it is important to normalize the shoulder width and rotation. Otherwise, we would immediately through all of our nearest neighbor classifications off since the distance metric would be comparing two gestures that were rotated differently or had different shoulder width, which could result in two very similar gestures having a large error, which would throw off the classification. As is, this problem is already run into solely because the gestures themselves vary so much in their sequences based on what was discussed above, so normalizing the shoulder width and the average direction makes these discrepancies easier to deal with in a nearest neighbors classifier.

Question 2: What is the best performance you can obtain on the test set by simply guessing the labels (i.e. the chance performance, without looking at the test data)? Explain your answer.

In the case of guessing randomly, we can assume we will get a success rate of 1/9 or approximately 11%. This is due to the fact that we have 9 possible gestures, and so guessing at random gives us a 1/9 chance of guessing the correct gesture.

Question 3: Explain why we can use the nearest neighbor algorithm only on fixedlength gestures. Implement the following function that normalizes each gesture sequence so that it consists of a fixed number of frames. Explain how you implemented your function.

Because our nearest neighbor algorithm works by comparing each gesture point by point, we need both gestures to have the same length. We are using L2 distance, which looks like $\sqrt{\sum(X_i + Y_i)^2}$. Here you can see that when comparing two gestures X and Y, we need to ensure that both gestures have the same length so we can compare them point by point.

To implement my normalizing function, I broke it up into three options:

1. Current Length = Normalized Length

In this case, I simply returned the gesture as is.

2. Current Length < Normalized Length

In this case, I had to add frames to the gesture. To do this, I first found the number of frames I needed to add by subtracting normalized length - current length. From here, I figured out the frames I should duplicate by defining a step as the current length divided by the number of frames to add. I then found the indices to

remove, which were defined by skipping 'step' number of frames at each increment, and returning the indices that the step stopped on. I then returned the gesture with those frames duplicated.

3. Current Length > Normalized Length

In this case, I had to remove frames from the gesture. To do this, I first found the number of frames I needed to remove by subtracting current length - normalized length. I also had to ensure I included the first and last frames. To do this, I automatically included 0 as an index to keep, and shorted my num_frames by 1. From here, I figured out the frames I should remove by defining a step as the current length divided by the number of frames to remove. I then found the indices to keep, which were defined by skipping 'step' number of frames at each increment, and returning the indices that the step stopped on. I added the last index, which brought me back up to the correct number of frames to keep. I then returned the gesture with those frames removed.

Question 4: Explain and justify the pros and cons of the nearest neighbor classifier for gesture recognition.

Nearest neighbor classification is useful because it is a very simple yet effective technique for classification problems. It does not involve much training or parameter tweaking to be highly effective. However, this also can be a negative because nearest neighbor relies solely on a distance metric, and must use a reliable distance metric in order to be effective. In this way it is not very robust to changes in input format. For example, if we did not normalize our inputs to have the same lengths, and if the gestures were not already pre-processed so that the shoulder width was always the same and the rotation was the same, our current distance metric and thus our nearest neighbor classifier would be much less accurate, since our simple distance metric would no longer be reliable.

Question 5: What is the accuracy when num_frames = 20 and ratio = 0.4? Try a few different values for the num_frames and ratio parameters and report the values that give you the best accuracy. For the default parameters you should get an accuracy of >70%.

NumFrames	Ratio	Accuracy
10	0.4	0.656
10	0.7	0.723
20	0.4	0.705
20	0.7	0.722
20	0.9	0.815
25	0.7	0.748
25	0.9	0.784
30	0.4	0.669
30	0.7	0.737
30	0.9	0.795
40	0.4	0.672
40	0.7	0.766
40	0.9	0.775

Table 1: Parameters

As seen in this table, having num_frames = 20 and ratio = 0.4 yields an accuracy of 0.728. The best parameters I found was num_frames = 20 and ratio = 0.9, which resulted in an accuracy of 0.815.

2 Decision Trees

Question 6: You'll see that the test accuracy changes each time. Why is this? What small modification stops the accuracy from changing?

The test accuracy changes because DecisionTreeClassifier has random_state by default set to None, which randomizes the classifying process and gives different accuracies each time. The sklearn.train_test_split() method has a set random_state variable which makes the split the same each time. To fix the accuracy, we

must also set random_state to an integer value in the DecisionTreeClassifier as well.

Question 7: Dissect the code and determine how we formatted the gesture data to input into the Decision Tree Classifier. Explain what our inputs, the features, X, and, labels, Y are and describe their shape (with specific numbers).

X has shape (270, 1188), while Y has shape (270,).

The features, X, are created from the sequences in the gesture sets. For each sequence in each gesture set, the code adds the list of frames in the sequence to a samples list, which X is assigned to. Thus, X is a list of all of the sequences in order for each gesture set. For each entry of X, corresponding to each sequence, we see that there are 36 frames in a sequence, and 33 values (corresponding to joints, etc) per frame. Thus, for each entry of X we have a list of length 1188 (33*36) showing each joint in each frame of that sequence. The labels, Y, are then the correlated label to the sequences that X contains. This means that the sequence at index i of X has the label at index i of Y. The Y list has length 270 (30 sequences per gesture, 9 gestures in the set) and each entry of Y has length 1 (one label per sequence). The X list has length 270 (30 sequences per gesture, 9 gestures in the set) and each entry of the list has length 1188 (36 frames per sequence, 33 points per frame).

The best ratio I found for train/test split was 0.9. This yielded an accuracy of 85%.

Question 8: Our model uses Gini Impurity, as seen in the DecisionTree constructor. Give its equation and describe what it measures.

The Gini impurity is a measure of the probability that we classify a datapoint incorrectly, if we were to classify it randomly based on the class distribution of the dataset.

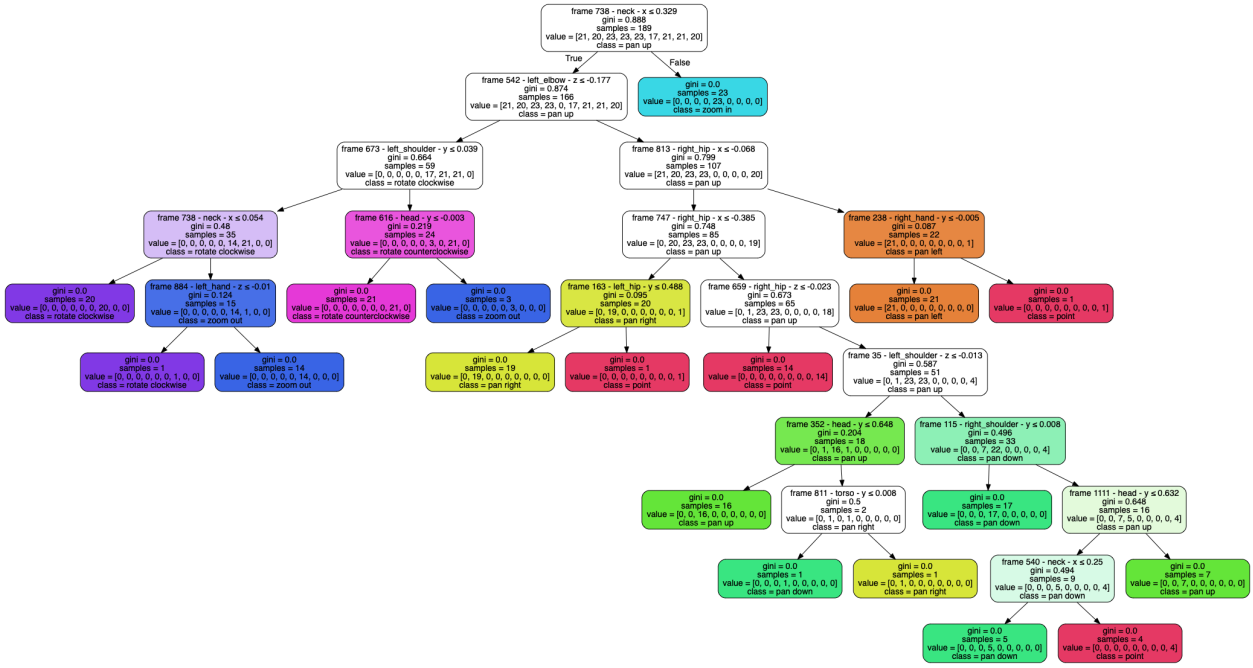
The Gini impurity has an equation as follows:

$$G(k) = \sum_{i=1}^{j} P(i)(1 - P(i)) \tag{1}$$

where P(i) is the probability of a certain classification i, and J is the total number of possible classifications.

Question 9: Inspect the tree. Report this tree's accuracy. What do the parameters gini, samples, value, class refer to? Why do all of the leaves have gini = 0?

Below you can see my decision tree from running decision_tree.py.



The tree's accuracy is 75.308.

At each node, the samples value represents the number of things to classify at each node, i.e. the amount of data points in the set at that node. The gini value represents the probability of an incorrect classification of the data given the classifications contained in the node. For example, we can see that in the class labelled zoom out, we have 15 samples total. If we look at its leaf nodes, we see that there is one class that has 1 sample, and another class with 14 samples. If we calculate the Gini impurity of the zoom out node, we get 1/15*(1-1/15)+14/15*(1-14/15) = 0.124, which we see is the gini value at the zoom out node.

The gini value at all of the leaves is 0 because gini impurity measures the probability of an incorrect classification of the data. By the time we get to leaf nodes, all of our data is the same classification. Thus, the probability of incorrectly classifying the data becomes 0 because all of our data is in the same classification, and we only have one possible classification at the leaf nodes to classify randomly into.

The values list is a list of length 9 where each index represents one gesture. The value at each index of the list gives a count of how many sequences were classified as that gesture (i.e index 0 is the 1st gesture, etc). For example, at the leaf nodes we get lists of all 0 except for one index, showing that all of the sequences have been classified as the same gesture at the leaf nodes. This also corresponds to a gini value of 0.

The class value is the gesture label of the mode of the values list. For example, when index 0 of values has the most counts, the node is given class pan left as this is the gesture that corresponds to index 0.

3 Recurrent Neural Networks

Question 10: LSTM networks require data to be formatted in a specific way. Analyze and describe how we format the gesture data for this network. Report the best validation accuracy you are able to attain and your values for each variable.

The X list has shape (270, 36, 33). The length of the first layer is 270 as this represents how many sequences are in the dataset. (9 gestures, 30 sequences per gesture). For each of these 270 indices, there is a list of length 36, which corresponds to the number of frames in each sequence. For each index of the 36, there is a list of 33 values, where each value corresponds to one point in the frame (i.e. the joints in the pose for each frame). The Y list has a shape of (270, 1), and is a list of 270 values with length 1 at each value, as the Y list simply represents the sequence label at index i corresponding to each sequence at the feature list index X[i].

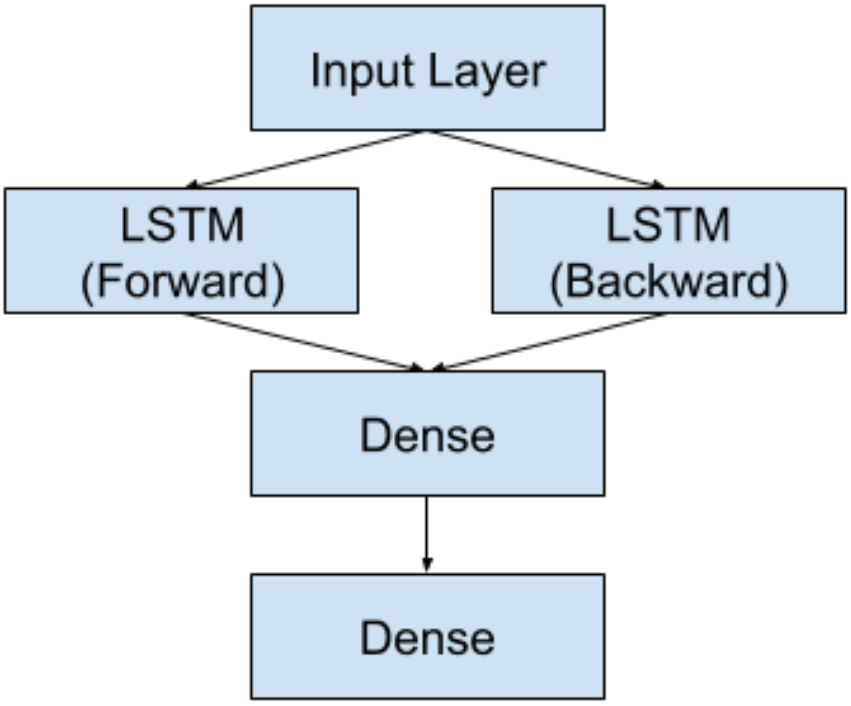
Batch Size	Epochs	Latent Dim	Loss	Acc	Val Loss	Val Acc
12	100	16	0.0221	0.9947	0.2880	0.9259
8	40	8	1.2111	0.5026	1.2521	0.4568
8	120	8	0.1719	0.9630	0.5133	0.8519
8	200	8	0.0342	0.9947	0.5375	0.8272
8	300	8	0.2165	0.9101	0.5791	0.8025
16	40	8	1.5486	0.3069	1.5407	0.2593
16	400	8	0.0955	0.9683	0.2510	0.9136
16	200	16	0.0551	0.9841	0.2659	0.9136
24	40	8	1.2492	0.5873	1.2832	0.4815
24	120	8	0.7278	0.7937	1.4011	0.6173
24	120	16	0.3139	0.8889	0.3940	0.8148
24	120	24	0.0071	1.0000	0.0926	0.9630

Table 2: Parameter Accuracy

The best validation accuracy found was 0.9630, using parameters of batch_size = 12, epochs = 120, and latent_dim = 24. This also gave us low loss (0.0071), high accuracy (1.00), and low validation loss (0.0926), ensuring that I was not over or underfitting the model.

Question 11: Draw a network diagram representing this new network (include the shapes of each input and output), similar to the original network drawing in Figure 4. Run your new model, report its performance as compared to the Unidirectional LSTM. (Hint: use the go backwards parameter in the LSTM constructor)

Below, you can see my network model for a Bidirectional LSTM.



Layer	Shape
Input	(?, 36, 33)
LSTM (Forward)	(?, 24)
LSTM (Backward)	(?, 24)
Dense	(?, 24)
Dense	(?, 24)

Table 3: Layer Shapes

When compared to the unidirectional model, the bidirectional model overall performed better. When using my best parameters of batch_size = 12, epochs = 120, and latent_dim = 24, the unidirectional model gave a validation accuracy of 0.9630. In the bidirectional case, I got a validation accuracy of 0.9753, where the loss was smaller, (0.0045) the accuracy was the same (1.0000) and validation loss was lower (0.0677).

Question 12: Provide the staff with feedback. Let us know what you think of the new Mini Project. What did you like about it? What did you not like about it? Were there any ambiguities that should be cleared up?

I enjoyed that this pset allowed us to see many different ways of solving classification problems. We were able to compare all of them and see the tradeoffs, in memory, efficiency, accuracy, etc. However, at times (especially with decision tree) I had a hard time figuring out how the code actually worked since I didn't get to do much implementation. Had I had more time I would have tried to complate the extra credit for this purpose, but I wish there had been more implementation opportunities.