**Peter Griggs**
**6.835 Mini-Project 2: Gesture Recognition using Kinect**

**Note:** There are two question 5's in the project handout, so I also included two question 5's in order.

**1)** I looked at different sequences in gesture 1 (pan right, or right hand moving right). Some of the differences I noticed between the sequences, excluding different number of frames, were: different ending position of the right hand, different amount of torso rotation while gesturing, and differing amounts of movement to the left before moving right.

We are using the location of each of the 11 joints in (x,y,z) space for the frames of the sequences as our training features. If the poses had different shoulder widths, this would be incorporated in the nearest neighbor classifier, because it might find significant differences between very similar gestures using a metric like euclidean distance to identify nearest neighbors. Additionally, average direction needs to be normalized to face the camera because this also skews classifying between frames. The locations of the features of a pose that faces the opposite direction is very different, so this would introduce errors.

**2)** The training set includes 9 gestures, each of which has 30 data points (the sequences). This is a uniform distribution of training data, so we would guess 30/270=1/9 that any given test sequence matches a label.

**3)** We can only use the nearest neighbor algorithm with a fixed-length of frames because the metric we use is euclidean distance, which can't be calculated correctly if there is a discrepancy between the dimensions of the vectors of frames for a gesture.

I implemented my function by trying to sample roughly every (num_frames/sequence_length) frame, because that is the most intuitive way. I referenced this stack overflow post (https://stackoverflow.com/questions/9873626/choose-m-evenly-spaced-elements-from-a-sequence-of-length-n) because it helped me debug my approach. The reasoning behind using math.ceil is that since we always sample the first frame, we won't sample the last frame without using ceil. For example, if we want to sample 3 elements from the array [1,2,3,4,5], using math.ceil will give us the subset [1,3,5], while we would get [1,2,4] without it.

**4)** One of the pros for using nearest neighbor classification for gestures is that it trains the model quickly relative to some other types of algorithms (like RNNs). Another one of the pros is that it is a very simple model to use and reason about. One of the cons is that because it is not very sophisticated, it is not great at differentiating between similar types of gestures (especially with the single nearest neighbor algorithm). Another con is that nearest neighbors stores all of the training data, which can be memory intensive.

**5)** The accuracy when num_frames=20 and ratio=0.4 is 0.74691. As shown below, the numbers that give me the best accuracy are when the parameters num_frames=30 and ratio=0.85, because the test accuracy is 0.944.

| num_frames | ratio | accuracy |
|---|---|---|
| 20 | 0.4 | 0.74691 |
| 20 | 0.5 | 0.82963 |
| 20 | 0.6 | 0.85185 |
| 20 | 0.7 | 0.79012 |
| 25 | 0.6 | 0.81481 |
| 15 | 0.6 | 0.84259 |
| 30 | 0.85 | 0.94444 |

**5)** When I run ‘python decision_tree.py’ multiple times, I noticed that the accuracies were different. When I looked at the source code/documentation for DecisionTreeClassifier, it says that features are always randomly divided at splits in the decision tree. In order to generate a repeatable splitting, you have to set random_state to a fixed value like 10, for example. When I tried setting random_state to an integer and re-ran `decision_tree.py`, I got the same accuracy.

**6)** X is an array where each element is an array that represents a sequence of frames. In the original representation of the data, each GestureSet has a list of 30 Sequences, each Sequence has a list of some number of frames, and each Frame is made of 11 points, or 33 values. In the decision_tree.py file, num_frames=36. Each sequence is 36*33=1188 and there are in total 9*30=270 sequences, so it makes sense that the shape of X is (270, 1188). Y is an array of labels that correspond to each sequence, so it makes sense that it has the shape (270), because that’s how many sequences there are.

To experiment with the best train_size, I set the random_state parameter so that I get deterministic sampling. Here are some of my accuracy results from changing the ratio:

| Ratio | Accuracy |
|---|---|
| 0.7 | 75.309 |
| 0.6 | 77.778 |
| 0.8 | 77.778 |
| 0.9 | 77.778 |
| 0.85 | 80.488 |
| 0.88 | 84.848 |

It seems that the best accuracy results from using roughly 88% of the total data for training.

**7)** The Gini Impurity criteria equation is given below. In words, it is a metric for how likely it is that a randomly chosen element is incorrectly classified given the distribution of our classes. For example, if we had a uniform distribution over our 9 gestures, we would have an I_g = 1 – 9*(1/81) = 8/9, which makes sense.

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} p_i \sum_{k\neq i} p_k = \sum_{i=1}^{J} p_i(1-p_i)$$

**8)** The accuracy of the decision tree is 81.818%. Gini is the Gini Impurity score at that node of the tree, generated by the distribution of labels at that node and the subset of the training data the tree has classified in each label. Samples refers to the number of training data points, which in our case are sequences, the decision tree is still considering at the specified node in the tree. Value is the distribution of samples over the labels that the tree has classified at that point in the tree. Class refers to the most common gesture that exists in the distribution.

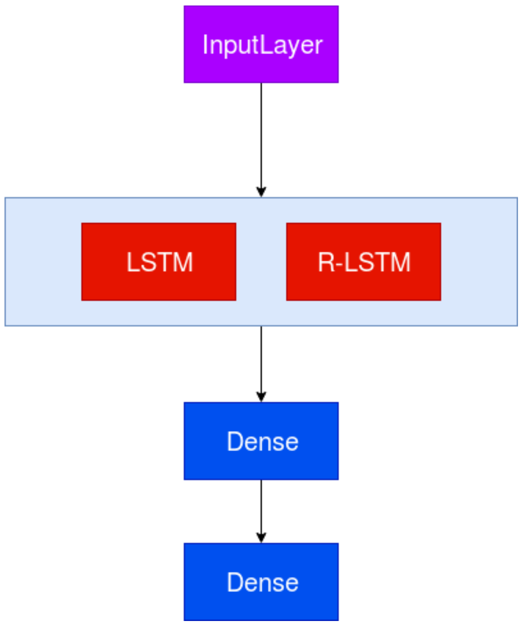
All of the leaves have gini=0 because the samples have been fully classified at the leaves. All of the samples at a leaf is in one label, so the Gini score is 1*0 (you will never incorrectly classify an item because there's one class).

**9)** The features, X, are an array of data representing the sequences. Each sequence is of shape (36,33), so the array has shape (270,36,33) because there are 270 sequences. The input layer needs to accept each sequence as training data, so it makes sense that we use the shape (36,33). Another part of formatting the data is that the labels are each put into their own arrays, so the shape of Y becomes (270,1) rather than just being a 1D array.

| Batch Size | Latent Dimension | Epochs | Validation Loss | Validation Accuracy |
|---|---|---|---|---|
| 12 | 16 | 100 | 0.3244 | 0.9136 |
| 8 | 16 | 100 | 0.7875 | 0.7654 |
| 14 | 16 | 100 | 0.5563 | 0.8642 |
| 24 | 16 | 100 | 0.2679 | 0.9383 |
| 12 | 8 | 100 | 1.1422 | 0.4198 |
| 12 | 24 | 100 | 0.2253 | 0.9383 |
| 12 | 20 | 100 | 0.3576 | 0.9012 |
| 24 | 24 | 100 | 0.3775 | 0.9136 |
| 12 | 16 | 200 | 0.4360 | 0.9136 |

The best validation accuracy I achieved was 93.83% with batch_size=12, latent_dim=24, and epochs=100. The validation loss was 0.2253, which was fairly low compared to other parameter settings.

**10)** The network diagram for the Bidirectional LSTM, note: R-LSTM means “reverse LSTM”



| Batch Size | Latent Dimension | Epochs | Validation Loss | Validation Accuracy |
|---|---|---|---|---|
| 12 | 16 | 100 | 0.2085 | 0.9506 |
| 8 | 16 | 100 | 0.2485 | 0.9259 |
| 14 | 16 | 100 | 0.1842 | 0.9259 |
| 24 | 16 | 100 | 0.2893 | 0.9259 |
| 20 | 24 | 100 | 0.0889 | 0.9630 |

The Bidirectional LSTM model seemed to do a bit better and was more consistent across changes in the parameter values, scoring over 90% on all of the parameter values I tried. The best validation accuracy I got was 96.30% with batch_size=20, latent_dim=24, and epochs=100 and the validation loss was 0.0889, which was lower than the highest accuracy run with the unidirectional model.

**11)** I liked that we got to try several different approaches on the same data and see the results. I also really liked how we used xbox kinect data, since that is something i’ve wanted to use in a project but have never gotten the chance to. One thing that could be improved is the structure of the data into GestureSet objects. I was pretty confused about what were our training samples and it took me a while to understand that the labels in GestureSet, Sequence, Frame were all duplicates. Overall, I really liked the lab but there could be some improvements with how the data is structured.