- Project2
- Due Date: **11:59PM, April 24**

Isolated digit recognition based on deep neural network/hidden Markov model

Note1: You can use existing the deep neural network libraries (pytorch and scikit-learn) for this project (for example, the instructor used scikit-learn). We installed both libraries.

Note2: This project requires an HMM implementation. People who cannot complete the HMM implementation in the project 1, please use the implementation made by the instructor. Please contact CAs.

1. Use the **same data** with the project 1
    1. Training data
    2. Test data

       Note that you can also access to the RAW WAV files (16kHz sampling)
    3. Data format
        1. Training and test data contain 2,464 and 2,486 utterances respectively.
        2. Each utterance has a unique id (e.g., ac_1a, ac_1b), and then actual features belonging to an utterance are followed by the corresponding id between the bracket "**[]**", as follows:

```
ac_1a  [
  -23.35901 -19.25983 6.819015 1.192464 1.073842 9.056857 -4.43707
13.45168 -2.058532 1.201817 13.34594 10.39188 19.14497 0.7117496 1.701741
1.048058 -1.132139 -2.300035 -0.0299722 -0.2512369 -3.788185 -4.024942
-4.57422 -3.920727 -4.042562 -1.762403 0.1584284 0.4702107 0.3236625
-0.3230287 -0.9805499 -0.7798319 -0.3273982 -1.54582 -0.7463068
-0.3776312 -0.1352869 -0.4284557 -0.5629961
  -22.46379 -16.336 9.509974 1.315522 4.676258 16.01371 -3.934596
5.259078 -16.13138 -10.392 -1.884554 -6.078195 10.01824 1.255656 3.066352
2.011334 -1.482856 -4.570226 -2.262345 -2.31138 -6.520723 -6.355066
-4.903043 -2.982224 -3.639628 -1.337538 -0.1939555 -0.3593823 -0.1568447
-0.0922939 -0.05669688 -0.7052898 0.09116052 0.08521891 1.433158 1.556499
1.694908 0.9933344 -0.1081609
  :
  :
    -23.82452 -16.94093 10.28893 3.670519 4.30678 16.45873 -0.3728658
12.118 7.065926 4.96498 12.72795 -1.431567 10.01824 -0.03938913
-0.1265938 0.01
485133 1.020298 0.7297202 0.3708503 0.04856908 0.6382608 1.61007
0.1387327 -0.1699445 0.3135568 1.261166 0.06409764 -0.06172538 -0.2362457
-0.16389
68 -0.02124497 -0.486647 0.4257887 0.07811558 -0.1541246 -0.0913877
-0.223996 -0.7058493 -0.2899268 ]
ac_1b  [
  -23.93669 -19.86685 3.774733 -3.180507 0.56567 9.032928 -2.525522
8.901489 -3.68621 3.336668 17.33731 0.6763301 10.17876 0.2232642
0.6983726 1.085476 0.4079107 -0.1208868 -1.017103 -0.2788303 0.4990832
1.54857 -0.02036728 -2.07991 -0.6725943 -0.06152797 -0.1132236 -0.1561639
0.1998773 0.3135434 -0.06890127 -0.2217057 -0.3023886 -0.4767102
-0.3915275 -0.4809449 -0.2955498 -0.7798449 -0.282258
  :
  :
```

3. After the "**[**", you can find a 39-dimensional feature vector per line. Each line correspoinds to a feature vector in a consecutive frame, and continued by "**]**".
    1. Utterance ID contains either 11 digit information {"1", "2", "3", "4", "5", "6", "7", "8", "9", "o", "z"}, e.g., utterance ID "ac_1b" includes "1" and this indicates the transription of "ac_1b" is "1"
    2. The other information (e.g., "ac_") in an utterance ID denotes a speaker id and data type, which are not used in this project.
4. Skelton script (`hmm_1digit_project2.py`, you can find this file in the project section of our Piazza page https://piazza.com/jhu/spring2020/en520666/resources):

1. We provide the skeleton script with the basic argument handling, data loader, and basic programming flow. You can only implement the core algorithms described below in the project.
2. **Please do not modify the argument options**

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('train', type=str, help='training data')
    parser.add_argument('test', type=str, help='test data')
    parser.add_argument('--niter', type=int, default=10)
    parser.add_argument('--nstate', type=int, default=5)
    parser.add_argument('--nepoch', type=int, default=10)
    parser.add_argument('--lr', type=int, default=0.01)
    parser.add_argument('--mode', type=str, default='mlp',
                        choices=['hmm', 'mlp'],
                        help='Type of models')
    parser.add_argument('--debug', action='store_true')
    args = parser.parse_args()

    # set seed
    np.random.seed(777)

    # logging info
    log_format = "%(asctime)s (%(module)s:%(lineno)d) %
(levelname)s:%(message)s"
    logging.basicConfig(level=logging.INFO, format=log_format)

    digits = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "z",
"o"]

    # read training data
    with open(args.train) as f:
        train_data = get_data_dict(f.readlines())
    # for debug
    if args.debug:
        train_data = {key:train_data[key] for key in
list(train_data.keys())[:200]}

    # read test data
    with open(args.test) as f:
        test_data = get_data_dict(f.readlines())
    # for debug
    if args.debug:
        test_data = {key:test_data[key] for key in
list(test_data.keys())[:200]}
```

```
    # Single Gaussian
    sg_model = sg_train(digits, train_data)

    if args.mode == 'hmm':
        model = hmm_train(digits, train_data, sg_model,
args.nstate, args.niter)
    elif args.mode == 'mlp':
        hmm_model = hmm_train(digits, train_data, sg_model,
args.nstate, args.niter)
        #TODO: Modify MLP training function call with appropriate
arguments here
        model = mlp_train(digits, train_data, hmm_model, nunits=
(256, 256), bsize=128, nepoch=args.nepoch, lr=args.lr)
```

3. **Please do not modify logging parts for the accuracy and likelihood**

```
logging.info("accuracy: %f", float(correct/total_count * 100))
```

   These loggings are used to check whether the program is working well or not

4. Use the debug mode when you test your scripts.

```
python submission.py --debug --mode mlp train_1digit.feat
test_1digit.feat
```

   Note that we increased the amount of training and test data in the debug mode from *100* to *200* utterances. If you continue to use your script, please make sure to use *200* utterances for the debug mode.

2. Data preparation for DNN training
   1. Prepare the HMM trained with the project 1
   2. Perform the GMM/HMM based Viterbi algorithm (made at the project 1) for the whole training data.

      If the digit of utterance u is "1", use the HMM for "1", and get an HMM state sequence $S^u = (s_t \in \{1, 2, 3, 4, 5\} | t = 1, \cdots, T^u)$ for each utterance u.
   3. Prepare unique HMM state IDs, e.g.,

      Digit "1": $j = \{1, 2, 3, 4, 5\}$

      Digit "2": $j = \{1, 2, 3, 4, 5\}$

      :

      :

      Digit "z": $j = \{1, 2, 3, 4, 5\}$

$\Rightarrow$

$l = \{1, 2, 3, 4, \ldots, 54, 55\}$

If you already use this representation for HMM state IDs, you can skip it.

4. Use this unique HMM state ID to convert the all state sequence obtained in the step 2, i.e.,

   $S^u = (s_t \in \{1, 2, 3, 4, 5, \cdots, 53, 54, 55\} | t = 1, \cdots, T^u)$

5. Perform the context expansion (3 left and 3 right context) for all feature vector sequences of the training data.

   $\mathbf{o}_t^{\text{new}} = [\mathbf{o}_{t-3}^\top, \mathbf{o}_{t-2}^\top, \mathbf{o}_{t-1}^\top, \mathbf{o}_t^\top, \mathbf{o}_{t+1}^\top, \mathbf{o}_{t+2}^\top, \mathbf{o}_{t+3}^\top]^\top$

   Note1) You require an edge processing for each utterance u: When $t < 1$, then $\mathbf{o}_t = \mathbf{o}_1$. Similarly, when $t > T^u$, then $\mathbf{o}_t = \mathbf{o}_{T^u}$

   Note2) Make sure the the number of dimensions become $39 \times 7 = 273$

6. Make a one big label vector and one big feature matrix by concatenating them for all utterances:

   - $\mathbf{s} = [s_1, s_2, \cdots, s_T]^\top \in \{1, 2, \cdots, 55\}^T$ where $T = \sum_u T^u = 238,122$
   - $\mathbf{O} = [(\mathbf{o}_1^{\text{new}})^\top, (\mathbf{o}_2^{\text{new}})^\top, \cdots, (\mathbf{o}_T^{\text{new}})^\top]^\top \in \mathbb{R}^{T \times 273}$

   Note: Make sure the dimensions of $\mathbf{s}$ and $\mathbf{O}$ are correct

7. Computer the HMM state prior distribution

   $p(l) = \frac{\sum_u \sum_{t=1}^{T^u} I(s_t = l)}{T}$ where $I(s_t = l)$ is an indicator function (1 when $s_t = l$, 0 otherwise)

3. DNN training (The instructor used scikit-learn)

   1. Set the DNN topologies (you can change your configurations if you want)
      1. Number of hidden layers = 2
      2. Number of hidden states = 256 for each layer
      3. Nonlinear activation function: Sigmoid function or ReLu function
      4. Randomly shuffle the training data (Make sure to shuffle $\mathbf{s}$ and $\mathbf{O}$ equally by preserving the input and output pair).
      5. Randomly extract 10% of the training data as validate data
   2. Perform the DNN training
      1. Optimizer: Adam (with a default value)
      2. Monitor the cross entropy for the training data and validation accuracy for each epoch
      3. Stop the training when the validation score starts degraded

   Check point: The DNN classification accuracy must be over 60% for the debug mode (70.3% with my implementation).

   Please print out the DNN classification accuracy with the following format:

```
Validation score: 0.703504
```

If you use scikit-learn, please specify `verborse=True`, then the validation accuracy is printed like above.

FYI, my setup for the MLP classifier in scikit-learn is as follows:

```
mlp = MLPClassifier(hidden_layer_sizes=(256, 256), random_state=1,
early_stopping=True, verbose=True)
```

4. Predict the most likely digit for each utterance by selecting the largest likelihood digit, i.e.,

   $\hat{\text{digit}} = \arg\max_{\{\text{"1"}, \text{"2"}, \text{"3"}, \text{"4"}, \text{"5"}, \text{"6"}, \text{"7"}, \text{"8"}, \text{"9"}, \text{"o"}, \text{"z"}\}} p(O^{\text{test utt}} | \Theta^{\text{dnn/hmm}})$

   The only change you need is in the Viterbi/forward algorithm:

   $p(\mathbf{o}_t | s_t = j, \text{digit}) \rightarrow p(s_t = l | \mathbf{o}_t) / p(s_t = l)$

   Be careful about the conversion of the HMM state ID given a digit (denoted by j) and the unique HMM state ID for all digits (denoted by l).

5. Compute the accuracy (# of correct digits / # of test utterances * 100) by using whole training data.

   Check point: The accuracy must be over 93% for the debug mode (95.5% with my implementation)

6. Final submission

   1. Prepare `submission.txt`, which only includes the accuracy result computed by DNN-HMM based speech recognition trained with **all training data**, and your name and leaderboard name as follows:

      ```
      NAME: <YOUR FULL NAME>
      LEADERBOARD_NAME: <LEADERBOARD NAME>
      MLP: 99.9
      ```

      Check point: The accuracy must be over 99% for all training data (99.9% with my implementation)

   2. Prepare your program (`submission.py`). We will confirm the two checkpoints above with the following commands. Please make sure that the following commands work correctly.

      ```
      python submission.py --debug --mode mlp train_1digit.feat
      test_1digit.feat
      ```

   We only accept the file name with `submission.txt` and `submission.py`. Please upload these files to our Gradescope project2 assignment.

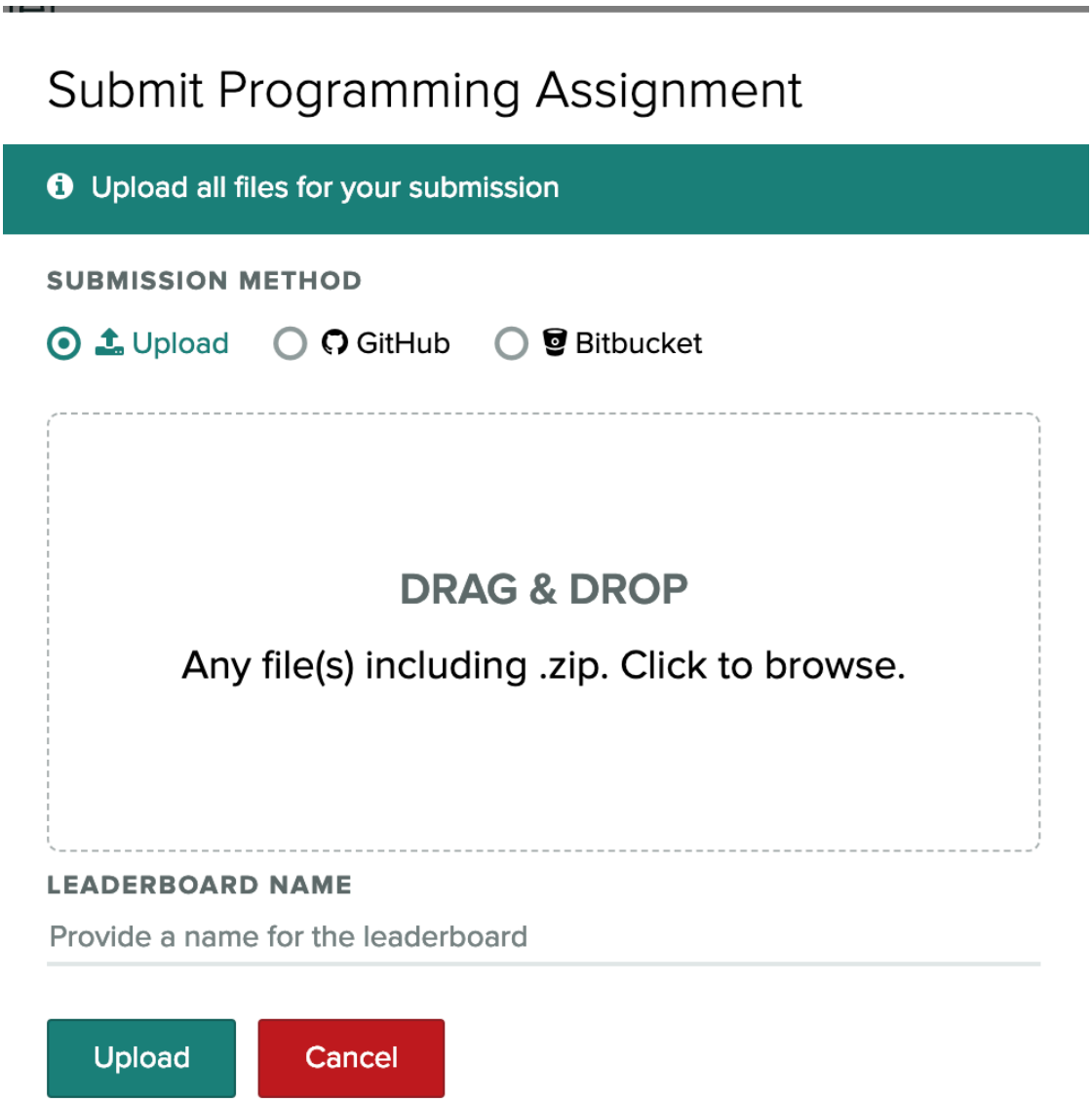
   3. Leaderboard

      1. The accuracy for the debug mode will be displayed in the leaderboard.

         1. You could tune the network topology or optimization configurations to get better

scores. You can post the result as many as possible.

2. You can use a nickname instead of your full name to *anonymize* your identity when you input the following LEADERBOARD NAME



3. Please correctly fill out your full name and LEADERBOARD NAME in `submission.txt` for CAs to identify the LEADERBOARD NAME and your full name.
4. We'll use leaderboard only for extra credits, like +3 for top 10%, +2 for next 10%, +1 for next 10%
5. People who have saved late days could have some advantages since they could post the result after most of people finish their submissions (This is our intension to give a bonus to such people).

Note: the accuracy you obtained with your local computer and leader board will be different due to random number or numerical issues.