COM4511/6511 Speech Technology

Practical Assignment: Isolated Spoken Digit Recognition

Overview

This individual assignment is about implementing, training and evaluating Hidden Markov Models (HMMs) for recognition of isolated spoken digits. A Viterbi-style HMM training and evaluation framework in Python is provided. The task is to complete the missing parts in the Python programs provided, conduct experiments on isolated digit recognition using the programs, and finally present the findings in a report. There is a bonus element as well, that allows to include Deep Neural Networks into the recognition process.

Deadline

The deadline for this work is **4pm Monday April 29th 2019** (week 10). Please see below for details of what to hand in, and how.

Resources

All the necessary files for this assignment are available on MOLE

- **data.zip**: Speech data containing isolated spoken digits from the TI-Digits corpus
- **hmm-viterbi.zip**: Python programs for HMM training and evaluation, to be completed
- **models.zip**: A set of pre-trained HMMs for validation of Viterbi decoding and training

You can use "`numpy`", "`scipy`", and "`scikit-learn`" (http://scikit-learn.org) for this assignment. The output distribution of each HMM state is represented by a Gaussian Mixture Model (GMM). The `GaussianMixture` package from "scikit-learn" is used for GMM parameter estimation and likelihood computation. *No other Python packages should be used for the main part of the practial* (just ask if you are not sure).

Task Description

The speech data is selected from the TI-Digits corpus and consists of isolated digits spoken by a large number of speakers of both genders. There are 11 digits in total: "zero", "oh", and "1-9".

The data is split into 2 exclusive sets: a training set and a test set. The training set is to be used for training the model parameters (100 utterance per digit). The test set should be used for testing only (60 utterances per digit). The list of utterances in each data set can be found in

- data/flists/flist_train.txt
- data/flists/flist_test.txt

The main programs are provided in the **hmm-viterbi.zip**. Once unzipped, the **hmm-viterbi** folder contains the following items (the incomplete files are listed in bold):

- **`hmm.py`** Defines a basic left-to-right no-skip HMM and employs Viterbi to find the most probable path through the HMM given a sequence of observations. Function `viterbi_decoding` is incomplete.
- **`hmm_train.py`** Performs Viterbi-style HMM training using `viterbi_decoding` from **hmm.py**. Function `viterbi_train` is incomplete.
- **`hmm_eval.py`** Performs evaluation of HMMs using `viterbi_decoding` from **hmm.py**. Function "`eval_HMMs`" is incomplete.
- `gmm_demo.py` Demonstrates the use of Gaussian Mixture Models (GMMs) for this task. This program serves as a baseline for comparison with the HMM-based recogniser. You should be able to run this program without any changes.
- `test_viterbi.py` Uses a set of pre-trained HMMs to test `viterbi_decoding` from **`hmm.py`**. For a test signal it computes the log-likelihood score and the most probable state sequence for the corresponding HMM and a mismatched HMM. The correct HMM would produce a higher score and a more reasonable state sequence.
- `speechtech/` A folder containing auxiliary functions needed for this task. You don't need to modify anything in this folder.

The Python scripts contain plenty of comments and the lines to be completed are clearly marked by "`====>>>> FILL WITH YOUR CODE HERE`". To help you complete the assignment, it can be split into three parts:

1. Viterbi Decoding;
2. Viterbi-style HMM Training; and
3. HMM Evaluation.

Part 1: Viterbi Decoding

The key part of this assignment is to implement the Viterbi algorithm for decoding the most probable state sequence through an HMM and computing the maximum likelihood for a given sequence of observations along the way.

Task: fill in the missing lines in function `viterbi_decoding` from **hmm.py**.

To help you validate your implementation of the Viterbi algorithm, a test script **test_viterbi.py** is provided:

1. Place the **data** folder in **`hmm-viterbi/`**.

2. Place the **`models`** folder that contains the file **`hmmfile_mfcc_10states_3mix`** (the pre-trained HMM set) in **`hmm-viterbi/`**.
3. The test script selects a signal from the test set and computes the log-likelihood scores and the most probable state sequences for the correct digit HMM and a mismatched HMM. The correct HMM would yield a better likelihood score and a more reasonable state sequence than the mismatched HMM.
4. Sample output:
   ==== Testing digit "1" (data/test/ap/1a.wav) ==== HMM "1": log_prob = -3891.77317788
   state_seq = [0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4 5 5 5 5 5 5 6 6 6
   6 6 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]
   ==== HMM "2": log_prob = -4156.49057536 state_seq = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4
   4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9
   9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]

Part 2: Viterbi-style HMM Training

In this part Viterbi training should be used to estimate the parameters of a strict left-to-right no-skip HMM with GMMs representing the output distribution from the train data set.

Task: fill in the missing lines in function `viterbi_train` from **hmm_train.py**.

Viterbi training requires the following steps to be taken. Note that multiple iterations of Viterbi training (Steps 2 and 3) are to be performed.

1. Initialisation: Adopt uniform segmentation to initialise state GMM parameters.

2. Accumulation: For each training utterance:

- Given an HMM and the sequence of feature vectors, find the most probable state sequence through the HMM using the Viterbi algorithm.
- The state-level alignment allows each feature vector to be assigned to one state. Collect all the feature vectors assigned to each state across all utterances.
- Count inter-state transitions (for estimation of transition probabilities).

3. Re-estimation: Update GMM parameters for states and the transition probabilities.

Most of the code is provided and you are asked to fill in the missing lines in Step 2.

Part 3: HMM Evaluation

Given a test utterance, the likelihood for each of the 11 digit HMMs needs to be computed and the one that has the maximum likelihood is chosen as the recognised digit.

Task: fill in the missing lines in function `eval_HMMs` from **hmm_eval.py**.

Bonus Part: Deep Neural Networks

For more advanced modelling neural networks can be introduced. The HMMs as trained in part 2 can be used with the algorithm of part 1 to obtain the training labels for NN data (note - the input to the deep neural network are standard features). Find out what would make sensible target labels, for using deep neural networks to obtain bottleneck features, or for recognition in so called hybrid configuration. Then train models and modify the training code (and recognition code if necessary) to work with neural networks, either in collaboration with or instead of GMMs.

- **train_nn.py**: script for training of (deep) neural networks using pytorch. Note that the topology (depth) and size of the networks can easily be modified.
- **eval_nn.py**: script for obtaining posteriorgrams of data given a specific NN. Note that the script can also be used to derive bottleneck features.

Practical Issues

- If Part 2 is not completed, use the pre-trained HMMs to do Part 3.

- **hmm_train.py** saves trained HMMs in the folder **models** in **hmm-viterbi** with a file name containing the parameter settings such as “hmmfile_mfcc_10states_3mix”.

- **hmm_eval.py** loads HMMs from an HMM file in the **models** folder. Make sure the parameters at the beginning of the script match the HMMs you want to use so that the correct HMM set is loaded.

- You are free to choose the feature type (MFCCs or FBANKs), the number of HMM states and the number of mixture components per state for your models. However, it is advised not to increase the number of states beyond 10 and the number of mixture components beyond 4 due to excessive training and testing time.

Report

A report explaining your results and findings should be produced in PDF or Microsoft Word format with a page limit of **4 pages**. You need to adopt a style commonly used for scientific writing, e.g. the IEEE ICASSP paper style. The templates are available at https://2018.ieeeicassp.org/papers/PaperKit.html#Templates

Only the recognition results on the **test set** should be reported. In general, your report should include the following sections:

- **Abstract**: A self-contained summary of the work so that readers can decide whether the paper is worth reading.
- **Introduction**: Describe the concerned problem, why it is important and what work has been done.
- The structure of the main body is up to you and it may include two or three sections explaining what you have done in this study.
- **Results and Discussion**: Discuss the results and explains why things turned out the way they did.
- **Conclusions**: A brief summary of the key points of the work.
- **References**: A list of suitable citations

Remember to use figures and tables where suitable. You may consider the following points in your report:

- Explain how the Viterbi algorithm works
- How was the HMM training performed? You should comment on the reasons for each step taken in training.
- How did the average log-likelihood on the training data change across each iteration?
- What are the testing procedures?
- Appropriate analysis of the recognition experiments outcome is to be performed. The analysis should include whether results are according to expectation and why that might be the case.
- By comparing your results with the GMM baseline results, explain what the advantage HMMs have over GMMs for speech recognition.
- Discuss results obtained with different features (MFCCs vs FBANKs), various numbers of states (e.g. 3 vs 10) and mixture components (e.g. 1 vs 3)

What to hand in

You should upload a **ZIP archive** containing your **source code** (three Python files: **hmm.py**, **hmm_train.py** and **hmm_eval.py**) and your **report** via MOLE. The course appears in the MOLE course list as *COM4511/COM6511 Speech Technology (SPRING 2017~18)*.

Please be aware that we scan the electronic submission of all students in order to detect plagiarism.

Marking criteria

30% of the module marks are assigned to this assignment.

Your work will be assessed according to the following criteria (out of 100%):

- Approach (code implementation, results obtained): 50%
- Presentation (code readability, report format, precision, visualisation of results): 20%
- Discussion (algorithm explanation, result analysis): 30%

Marks and feedback will be returned to you by Monday 21st May 2018.

Late work

Late work will be penalised according to standard Departmental procedure (a penalty of 5% per day late; work will be awarded a mark of zero if more than 5 days late). The hand-in date is taken to be the date on which the electronic files were uploaded to MOLE. See:

http://www.dcs.shef.ac.uk/intranet/teaching/public/assessment/latehandin.html