Instructions: This assignment is intended to introduce you to signal processing to the extent that it is used for typical speech recognition methods. Additionally, the quality of these features will be tested with the implementation of one of the oldest methods for speech recognition: Dynamic Time Warping (DTW). Please ask for help if you feel stuck!
Please submit answers to all underlined questions below in the writeup as a PDF file via the course Canvas site. The tex file used to create this PDF has also been provided so that you can enter your answers directly into this document. Additionally, please run through and execute all of the cells of the ipython notebook (including the parts you added which are marked by `# TODO`) and submit the notebook via Canvas.
Lateness policy: Late submissions submitted up to Thursday 4/30/20 7:00 pm (the "late deadline") will receive a 20% reduction in credit. Submissions that are later than this will receive some non-zero credit, but we cannot guarantee how much. We cannot guarantee that we will carefully grade or give feedback on submissions after the "late deadline". In addition, you have four free "late days" that you can use throughout the term to extend homework (not project) deadlines without penalty. If you are using any of your "late days" for this assignment, mention that in the comment while submitting this assignment. You should state how many "late days" you are using and how many remaining "late days" you have. The number of late days used for any given homework must be an integer.
Collaboration: You are encouraged to discuss assignments and any aspect of the course material with others, but any material you submit (writeup, code, figures) should be produced on your own.

0. Please fill out this questionnaire. This is purely to help us calibrate assignment load and let us know what may need to be made clearer in class. Your responses will receive a small amount of credit, independent of the actual answers.

   (i) Did you collaborate on this assignment, and if so, with whom?
   No.

   (ii) Approximately how many hours did this assignment take to complete?
   8 - 10 hours.

   (iii) On a scale of 1 to 5 (where 1 = trivial and 5 = impossible), what was the difficulty of this assignment?
   4. Just right.

   (iv) On a scale of 1 to 5 (where 1 = useless and 5 = essential), how useful has this assignment been to your understanding of the material?
   5. I've implemented the String Edit Distance algorithm before, so learning to do DTW in a similar fashion is fun.

1. Mel-frequency Cepstral Coefficients (MFCCs)

   (a) This part allows you to experiment with MFCCs, and introduces you to code to perform feature extraction in python.

   DFT benchmark report

   ```
   with a window size of 128 and a step of 32
   1.21 ms ± 338 µs per loop (mean±std. dev. of 7 runs, 50 loops each)
   1.45 ms±348 µs per loop (mean±std. dev. of 7 runs, 50 loops each)


   with a window size of 512 and a step of 32
   12.1 ms±1.18 ms per loop (mean±std. dev. of 7 runs, 50 loops each)
   3.04 ms±196 µs per loop (mean±std. dev. of 7 runs, 50 loops each)


   with a window size of 2048 and a step of 32
   210 ms±28.9 ms per loop (mean±std. dev. of 7 runs, 50 loops each)
   12.1 ms±5.61 ms per loop (mean±std. dev. of 7 runs, 50 loops each)
   ```

   We observe that as window size increases, FFT becomes increasingly more time efficient relative to DFT.

   (b) Follow the instructions given in Part 1 of the `hw2.ipynb` and answer the questions below.

   - What does the first cepstral coefficient $c[0]$ represent? (i.e. can you describe it in words, by examining the equation for computing the cepstrum given in the lecture slides)
     Optionally (for extra credit):
     What about the second cepstral coefficient $c[1]$?
     Answer: $c[0]$ represents the average energy of the input signal. $c[1]$ reflects how balanced the distribution between low-frequency and high-frequency signal is.
     Source: `https://www.researchgate.net/post/why_most_of_reasearchers_consider_13_MFCC_coefficients for_analysis_of_speech_or_images`

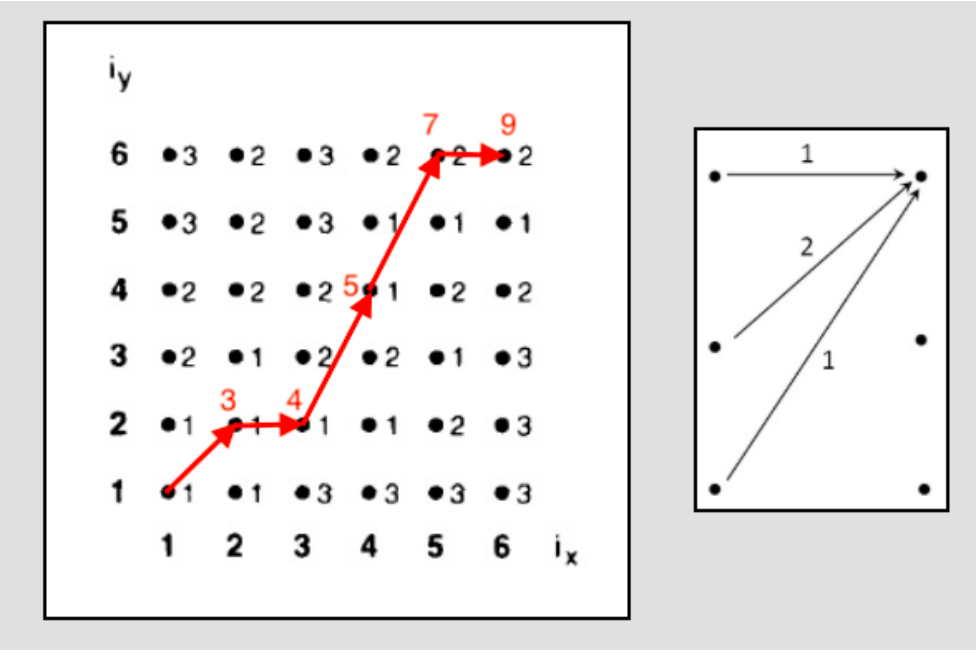
   - How does the recovered signal compare to the original recording? Does the difference make sense?
     Repeat for smaller numbers of cepstral coefficients by lowering 'ncoeffs'.
     What's the smallest number such that the re-synthesized utterance is still intelligible?
     The recovered signal has a mechanical sound to it. It's missing any indication of tone or speaker information. This makes sense since our goal is to capture solely the acoustic features and hence our pipeline suppresses the auxiliary information.
     Even when ncoeffs is as small as 3, I still found the re-synthesized utterance quite intelligible, especially the digit three. The utterance becomes pretty non-intelligible when ncoeffs is 2 and we only have c[0] and c[1].

   - Which types of speech sounds do you expect to be better represented by long-window MFCCs and which by short-window MFCCs? For purposes of this question, short-window MFCCs are ones generated with a window size of 64 samples, and long-window MFCCs are generated with a 1024-sample window. The long-window version is the standard one used in speech recognizers. Optionally (for extra credit): Can you suggest a way of using this information to improve the performance of a speech recognizer?

     From the lecture slides, MFCC windows should be short enough so that we can assume stationarity in the window and long enough for good frequency resolution. Therefore, I expect short utterance in bursts (such as the pronunciation of single digits or even single phones) to be better represented by short-window MFCCs. My hypothesis is partly based on the observation that we are using a (not too large) window size of 256 samples for our digit-recognition task. As for long-window MFCCs, I expect them to better represent long, continuous speech flow like regular conversations.
     Optional: I'd suggest extracting both long-window and short-window MFCCs and combining them into new MFCC features. This may be especially useful if the speech flow consists of both whole sentences and short utterances like people spelling out names or phone numbers.

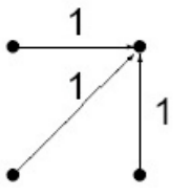
2. Dynamic Time Warping (DTW):

   (a) Complete the DTW exercise shown below using the following move set. Find the minimum distance, and the corresponding path, through the grid of frame distances using the move set below. Submit the marked-up grid with the best path (or one of the best paths, in case of ties) highlighted and minimum distance marked. **Answer: I found the minimum distance to be 9.**



   (b) For this part you will use a basic DTW-based recognizer to do single digit recognition. You will use a data set consisting of 100 training utterances and 100 test utterances of isolated digits (10 each of 0-9, where "0" is always pronounced "oh" and not "zero"). The data and scripts are to be downloaded from the canvas site.

      i. Finish the implementation of the 'dtw' function by implementing the recursion step (marked `# TODO`) for the move set introduced in class:



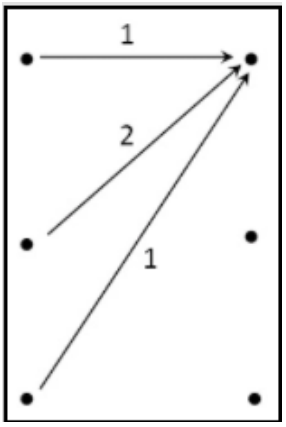
      ii. Additionally, experiment with (at least) one of the following:
         - altering the 'dist' function
         - changing the move set
         - increasing the number of templates
         - coming up with your own extension!

      iii. Describe in 1-2 paragraphs what you did, the results you obtained, and any other comments about your experiments (e.g. the trade-off between performance and efficiency, types of errors, etc.).

      Extra credit for the best DTW-recognizer. Note that this is a "cheating" experiment in that we are tuning the recognizer on the test set. In a "real" experiment, we would have a separate development set for tuning, and only test on the test set after fixing the recognizer based on development set performance.)

Each training cycle takes about 5 minutes on my Mac. My best result is 89% accuracy, obtained with the move set in i. and Manhattan distance.
I achieved 87.00% accuracy with the move set introduced in class paired with Euclidean distance.
In addition, for ii. altering the 'dist' function, I tried using Cosine distance (scipy.spatial.distance.cosine),

which yielded a 86.00% accuracy. I also experimented with Manhattan distance (scipy.spatial.distance.city and obtained a slightly improved 89.00% accuracy. This is a little unexpected and I'd love to learn the intuition during future Office Hours.
For ii. changing the move set, I implemented another version of DTW using the move set in 2 (a).



This move set was more difficult to implement because we don't have an option to move vertically up: This means that all cells above the first row and in the first column are invalid intermediate destinations. In my implementation, I made their cost float('inf'). I confirmed the correctness of my implementation by testing it on the DTW by hand exercise in (a). However, in actual training, for about 10 examples, backtracking ended up in one of the unreachable first-column cells and could not get back to the origin. I spent about an hour debugging but couldn't resolve it. My guess is that there are close ties among the costs and the loss of floating point precision and my use of np.isclose led to corner cases. As an engineering solution, whenever this happens, I return a float('inf') distance so that during testing, a buggy example will never become the top-pick digit. I also tried another approach: using try-expect to skip over buggy examples during training and testing. Both approaches might non-trivially hurt my accuracy and both yield an accuracy of about 81%, the lowest among all my approaches. My takeaway from this experiment is that designing move sets require good understanding of the data and advanced domain knowledge.
I tried increasing the number of templates to 2, but the training was taking more than 10 minutes so I don't have any result to report.

3. When spelling a word over a (land-line) telephone, we often find ourselves disambiguating letters; e.g., "S as in Sam, F as in Frank, B as in boy, D as in dog...". This is more necessary to disambiguate some letter pairs than others. S and F are a particularly difficult pair. (Note that we don't need to do this when talking face-to-face, even when not looking at our interlocutor.)
Why is it necessary to do this disambiguation, and why are S and F particularly difficult?

Answer: We mentioned in the lecture that land-line telephone service usually has a narrow frequency range. Wikipedia says it's about 300Hz to 3.4kHz and our lecture says it's below 4kHz. This means that some of the higher frequency sound will get cut off by the land-line phone. S and F are particularly difficult to distinguish because both are fricatives that span 4 - 7 kHz.

4. When listening to high-pitched singing, it can be harder to distinguish vowels than in low-pitched singing. (For our purposes, we can think about singing as identical to speaking but at specific pitches dictated by the musical score.) For example, if a singer produces an [iy] and an [aa] on a "high C", the two vowels may be indistinguishable. Note that the fundamental frequency of a "high C" is close to 1000Hz.
Give one explanation for this effect. Note: The spectrum of a vowel at fundamental frequency F will consist roughly of very high values at multiples of F and almost 0 at other frequencies.

Answer: We are able to distinguish between different vowels in a lower-pitch setting because their three formants on the spectrogram are very different. For instance, according to the MIT plot of vowel frequencies of female speakers, the three formants for [iy] in Hz are roughly 600, 2200, 2900, whereas for [aa], 800, 1300, 2600. According to Wikipedia, humans are most sensitive to sounds between 1 kHz and 4 kHz. These formants happen to fail within the range humans are most sensitive

to, and hence humans can easily distinguish vowels in their normal formants. However, when vowels are produced at a high fundamental frequency F and only consist of very high values at integer multiples of F, the human ears won't be able to easily pick up the different formants that are used to distinguish vowels in more common frequency settings.

5. Speech recognizers often (but by no means always) perform better on male voices than on female voices. There are many possible reasons for this, some of them having to do with the different typical vocal tract properties of men and women. (Optional, extra credit) Explain how male vs. female vocal tract properties may cause this effect.

   Answer:
   Reference: Ankita's Office Hours, `https://www.theregister.co.uk/2018/03/14/voice_recognition_systems_are_naturally_sexist`**,** `https://www.sciencedirect.com/topics/medicine-and-dentistry/vocal-tract` **This question is related to the last one in the sense that vowels may become indistinguishable at a higher fundamental frequency.**
   Females usually have a higher fundamental frequency than males, and this is partly because females usually have shorter vocal tracks and smaller vocal folds than males. The smaller vocal folds produce more vibrations and higher pitch values. As an analogy, larger musical instruments usually produce lower-pitched sounds while smaller instruments produce higher-pitched sounds. Moreover, relating to Q4, if the speech recognizer have frequency bandwidth cut-offs like land-line telephones, it might be more difficult for them to pick up and disambiguate high frequency fricatives like S and F produced by female speakers.