IEMS5726 Data Science in Practice (2025-26)
Assignment 2
Expected time: 6 hours

Learning outcomes:
1. To understand the pipeline in data preprocessing and data representation 2. To practise data preprocessing and data representation for numeric, categorical, natural language, image and audio data

Instructions:

1. Do your own work. You are welcome to discuss the problems with your fellow classmates. Sharing ideas is great, and do write your own explanations.
2. If you use help from the AI tools, e.g. ChatGPT, DeepSeek and Gemini, write clearly how much you obtain help from the AI tools. No marks will be taken away for using any AI tools with a clear declaration.
3. All work should be submitted onto the blackboard before the due date.
4. You are advised to submit the following two items.
    a. A .pdf file containing the answers of the written part.
    b. A Assignment2_1155xxxxxx.py file storing all your programs for the five problems. (1155xxxxxx refers to your student ID)
5. Do type/write your work neatly. If we cannot read your work, we cannot grade your work.
6. We will grade your work based on Anaconda Python version 1.13.
7. The sample output in the specification is run based on Windows. Your output may be different if you run the programs on Mac or Linux machines.
8. If you do not put down your name, student ID in your submission (both .pdf and .py), you will receive a 10% mark penalty out of the assignment 2.
9. Due date:
    a. Session A: 5th February, 2026 (Thursday) 23:59
    b. Session B: TBC April, 2026 (Saturday) 23:59

Short questions (25%)
Answer all questions.

1. You are developing an AI-powered chatbot system to enhance customer service efficiency and user experience on Taobao's platform.

   a. Suggest two main objectives in a chatbot system for customer service. (2%)

   b. Suggest two main differences between LLM and NLP. (2%)

   c. What are word embeddings? (1%) How can word embeddings be used in NLP? (1%)

   d. There are three commonly used word embeddings, including (i) count-based embedding, (ii) dense embedding and (iii) pretrained embedding. Suggest one tool for each of the three word embeddings. (3%) Explain briefly how the embeddings are constructed by these three methods. (3%)

   e. When we download a pre-trained model, what else do we need to download together with the model? (1%)

   f. In a transformer, there are many layers of encoders and decoders. Why do we need that many layers, instead of a single encoder with a single decoder? (2%)

   g. When we build the chatbot system, which type of the LLM do you prefer, encoder-only, decoder-only or transformer? Explain your choice. (2%)

   h. What is fine-tuning in the context of LLMs? (2%)

   i. Suggest and explain three metrics that you can evaluate the performance the chat bot. (6%)

Remarks: In assignment 2, you need to fill in some explanations and outputs for question 6 in your written part.

Practical problems (75%)
Work on the following five problems in a single .py file.
If your Python program cannot be run, you will receive no scores for the problem.
Put down your student ID as a comment in your first line of the program.

2. Write a function that splits the dataframe into training and validation set and testing set in the ratio (1-t):t; and splits the training and validation set in the list of k dataframes with size difference at most 1. (15%)
   This problem covers the split of datasets.

   Code:

```
# <Your student ID>
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# Problem 2
def problem_2(df, k=5, t=0.2):
    list = []
    # write your logic here, test is a dataframe
    test = 0


    return list, test
```

   Execution:

```
> df = pd.read_csv("problem2.csv")
> list, test = problem_2(df, k=5, t=0.2)
> for item in list:
>   print("Segment: ", item.shape)
Segment:  (166, 6)
Segment:  (166, 6)
Segment:  (165, 6)
Segment:  (165, 6)
Segment:  (165, 6)

> print("Testing: ", test.shape)
Testing:  (207, 6)
```

3. Write a function that performs one-hot encoding, based on the input filename and the label of the column. (15%)
This problem covers basic functions and Pandas operations.

Code:

```
# Problem 3
def problem_3(filename, column_label):
    # write your logic here, df is a dataframe instead
    df = 0

    return df
```

Execution:

```
> df = problem_3("problem3.csv","color")
> print(df)
   label  color  blue  green  red
0      1    red     0      0    1
1      2    red     0      0    1
2      3  green     0      1    0
3      4  green     0      1    0
4      5   blue     1      0    0
5      6  green     0      1    0
6      7   blue     1      0    0
```

4. The file, `TRAIN.csv`, specifies the names of the 250 wav files inside the folder `TRAIN`. Write a function that loads all the 250 audio files into a 3D tensor using torchaudio. Convert all the wav files into its mel spectrogram in decibel through truncating or padding zero to the desired length (= 100000). Return the tensor containing 250 × mel spectrograms in decibel. (15%)
This problem covers audio signal processing.
Data source: https://www.kaggle.com/datasets/imsparsh/audio-speech-sentiment

Code:

```
# Problem 4
import os
import torch
import torchaudio
from torchaudio import transforms
def problem_4(path):
    # Don't touch the settings below
    desired_length, n_fft, hop_len, n_mels, top_db = 100000, 1024,
        256, 32, 80
    list_of_mel_spec_db = []
    # retrieve the list of files under path
    file_names = os.listdir(path)

    for file in file_names:
        # write your logic here: load audio file


        # get the current length of the waveform
        current_length = waveform.size(1)  # Size (channels, length)

        # write your logic here:
        # truncate the waveform, or pad the waveform with zeros


        # write your logic here: define mel spec with the settings
        mel_transform = transforms.MelSpectrogram(
            sample_rate= 1,
            n_fft= 1,
            hop_length= 1,
            n_mels= 1
        )

        # write your logic here: apply the transform to the waveform
        mel_spec = 1

        # write your logic here: convert to decibels with settings
        mel_spec_db =
            transforms.AmplitudeToDB(top_db= 1)(mel_spec)

        list_of_mel_spec_db.append(mel_spec_db)

    # convert the list to a tensor
    # assume all mel spec have the same shape
    mel_spec_tensor = torch.stack(list_of_mel_spec_db)

    return mel_spec_tensor
```

Execution:

```
> tensor = problem_4("TRAIN")
> print("Type of tensor:", type(tensor))
Type of tensor: <class 'torch.Tensor'>
> print("Shape of tensor:", tensor.shape)
Shape of tensor: torch.Size([250, 2, 32, 391])
> torch.save(tensor, "problem_4.pt")
```

5. This problem is the continuation of problem 4. Write a functions that train an autoencoder with the following architecture with the 250 × mel spectrograms in decibel. Return the MSE for the 20 epochs. (15%)
   - Input: d = 2 * 32 * 391
   - Encoder: d → 8224 → 2056 → 1024 → 512 → 256
   - Latent Space: 256
   - Decoder: 256 → 512 → 1024 → 2056 → 8224 → d
   - Output: d

   Due to different seeds and different OS, the output will be different.
   This problem covers autoencoders.

   Code:

```
# Problem 5
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn, optim

class MyDataset(Dataset):
    def __init__(self, tensors):
        self.tensors = tensors
    def __len__(self):
        return len(self.tensors)
    def __getitem__(self, idx):
        return self.tensors[idx]

# write your autoencoder structure here
class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()
        self.encoder = nn.Sequential(

        )

        self.decoder = nn.Sequential(

        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded, encoded

def problem_5(tensor_file):
    loaded_tensor = torch.load(tensor_file, weights_only=True)
    dataset = MyDataset(loaded_tensor)
    dataloader = DataLoader(dataset, batch_size=25, shuffle=False)

    model = AE()
    loss_function = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-3,
        weight_decay=1e-8)
    epochs = 20
    losses = []
    device = torch.device("cuda" if torch.cuda.is_available() else
        "cpu")
    model.to(device)

    for epoch in range(epochs):
        loss1 = []
        for audio in dataloader:
            audio = audio.view(-1, 2 * 32 * 391).to(device)
            reconstructed = model(audio)[0]
            loss = loss_function(reconstructed, audio)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            loss1.append(loss.item())

        losses.append(sum(loss1) / len(loss1))

    return losses
```

Execution:

```
> losses = []
> losses = problem_5("problem_4.pt")
> i = 1
> for loss in losses:
>     print(f"Epoch {i}/20, Loss: {loss:.6f}")
>     i += 1
Epoch 1/20, Loss: 2201.029639
Epoch 2/20, Loss: 2192.524243
Epoch 3/20, Loss: 2192.817383
Epoch 4/20, Loss: 2192.949707
Epoch 5/20, Loss: 2192.982861
Epoch 6/20, Loss: 2192.745239
Epoch 7/20, Loss: 2192.657373
Epoch 8/20, Loss: 2192.639600
Epoch 9/20, Loss: 2192.632593
Epoch 10/20, Loss: 2192.632910
Epoch 11/20, Loss: 2192.633960
Epoch 12/20, Loss: 2192.634888
Epoch 13/20, Loss: 2192.636426
Epoch 14/20, Loss: 2192.637842
Epoch 15/20, Loss: 2192.639136
Epoch 16/20, Loss: 2192.641162
Epoch 17/20, Loss: 2192.642407
Epoch 18/20, Loss: 2192.643140
Epoch 19/20, Loss: 2192.643213
Epoch 20/20, Loss: 2192.643506
```

6. This problem is the continuation of problem 5. Write a functions that train another autoencoder with the 250 × mel spectrograms in decibel. Return the MSE for the 20 epochs. There are several limitations. (15%)
   - The number of epochs remains 20.
   - The size of the latent space remains 256.
   - The loss (MSE) at the 20-th epoch has a significant improvement.

   In the written part:
   - Explain how you achieve a significant improvement
   - Include the outputs of your 20 epochs

   This problem covers autoencoders.

   Code:

```
# Problem 6
def problem_6(tensor_file):
    # write your logic here
    losses = []


    return losses
```

   Execution: Output is skipped, as you are going to fill in your output in the written part.

```
> losses = []
> losses = problem_6("problem_4.pt")
> i = 1
> for loss in losses:
>     print(f"Epoch {i}/20, Loss: {loss:.6f}")
>     i += 1
```

More to know about question 4 - 6:

The primary focus on the audio dataset is to perform the classification task on the sentiment (positive, neutral or negative). Using the classic machine learning approaches, audio features (i.e. MFCCs) are extracted from the files and applied for various classification algorithms (e.g. Random Forest, Support Vector Classifier, … etc).

After problem 5 and 6, you can use audio features from the autoencoder (i.e. the latent space retrieved from the `encoded` in the autoencoder) to run the classification algorithms. In general, the results using the latent spaces achieve a higher F-measure compared to using the MFCCs.

< End of Assignment >