Due to the large size of our input data, we use SOC compute clusters to run the following tasks. As we split the tasks among ourselves and each task is run on different servers, we provide the python files and jupyter notebooks used for each step in this GitHub repo.

The following documentation provides a step-by-step guide on how to replicate our results using this repo. As running each file can take hours given the large data size, to check code accuracy, we suggest random sampling a small portion of the original input and see whether the codes are error-free.

The output files generated by each step is not uploaded to GitHub due to size limitation. Hence, we provide the link to those files stored in GDrive.

Step 1: EDA

- file path: *Data Analysis/EDA.ipynb*
- input requirement
    - download the full review csv at GDrive, save it to your directory
    - replace the csv reading command with the data path on your device
      df = pd.read_csv('your_path/yelp_review.csv')
- run the cells in the notebook to perform EDA

Step 2: Data Preprocessing

- preprocessing (non-English reviews removal, data split, lemmatization, tokenization)
    - file path: *Data Preprocessing/detect_language.py, Data Preprocessing.ipynb*
    - input requirement
        - download the full review csv at GDrive, save it to your directory
        - replace the csv reading command with the data path on your device
          df = pd.read_csv('your_path/yelp_review.csv')
    - run python detect_language.py first to detect the language of each review
    - run the cells in Data Preprocessing.ipynb for the remaining preprocessing steps
    - the generated train.csv, val.csv and test.csv can be accessed on GDrive

- preprocessing (label assignment, undersampling)
    - file path: *Data Preprocessing/undersample.ipynb*
    - input requirement
        - use the 3 csv files generated from the previous preprocessing step, which can be downloaded using the respective link: train.csv, val.csv and test.csv can be accessed on GDrive
    - create a folder named data in your directory for storing the output
    - run the cells in undersample.ipynb for the remaining preprocessing steps
    - the generated processed_train.csv, processed_val.csv and processed_test.csv can be accessed on GDrive

Step 3: Text Representation

- file path: *Embedding/vectorizer.py, doc2vec.py, sbert.py*
- input requirement
    - data/train_processed.csv, val_processed.csv and test_processed.csv generated from undersampling.ipynb is used as input
    - replace variables train_path, val_path and test_path with the directory on your device based on where you store the previous output
    - create a folder called text_processing and 3 subfolders with names including cool_processed_text, funny_processed_text and useful_processed_text to save the output
- change the category variable accordingly based on the dimension of
- run python vectorizer.py to generate representations using the three vectorizers
- run python doc2vec.py to generate word embeddings encoded using doc2vec
- run python sbert.py to generate word embeddings encoded using SentenceBERT
- file output
    - the 3 python files output text representation using different methods
    - due to space limitation, output files are not uploaded to GitHub
    - they can be accessed on GDrive

Step 4: Classifier Training

- file path: *Model_Running/**
- input requirement
    - data/train_processed.csv, val_processed.csv and test_processed.csv generated from undersampling.ipynb is used as input
    - text_processing/* generated from Step 3 is used as input
    - except for ALBERT_fine_tune.py and BERT_fine_tune.py where only train.csv, val.csv and test.csv are used as input
- change the input file path to your directory to run the files
- run model_run_{useful/funny/cool}_npy.py to train classifiers using embedding representations generated by doc2vec or SentenceBERT
    - change the model variable accordingly to use different embedding
- run model_run_{useful/funny/cool}_npz.py to train classifiers using text representations generated by the 3 vectorizers
    - change the model variable accordingly to specify the corresponding vectorizer
- run ALBERT_fine_tune.py to fine tune ALBERT using only the reviews and the vote label
- run BERT_fine_tune.py to fine tune BERT using only the reviews and the vote label