IDMIL: An alignment-free interpretable deep multiple instance learning (MIL) for predicting disease from whole-metagenomic data

The python scripts used in the implementation of IDMIL is documented here with their brief description, dependencies, parameters and expected output.

Preprocessing

We perform some rudimentary preprocessing on the FASTQ files containing raw DNA sequences. For this purpose, we use Fastp an efficient FASTQ processing tool. We removed nucleotides with quality score less than 20. We also removed sequence reads whose 30% of nucleotides are with quality score less than 20. Any sequence with length less than 200 nucleotides is also removed. We merged the end-pair sequences to get a single FASTA file per sample.

The python scripts expect each training sample as a single FASTA file in a directory. Similarly, each test sample should be represented as a single FASTA file in a separate directory. There should be a text file recording the class labels of all the samples. Each new line in the text file should contain a sample filename and its class-label (0 or 1) separated by a tab.

Vocabulary Building and kmer embedding

Step 1: Build vocabulary and prune using TF-IDF

- **vocabulary.py**
    - ***Dependency***:
        - *PyTables*: An opensource tool providing persistence hdf support. PyTables support calculation on persistent data avoiding memory overflow for very large data (i.e., large value of kmer).
    - ***Purpose***: This script extracts all the kmers of a user-provided length k from the FASTA. It then calculates the TF-IDF score for each kmer and removes a user-specified amount of kmers based on this score.
    - ***Arguments***:
        - *-input* : provide the path to the directory containing training data.
        - *-output* : provide the path to the output directory.
        - *-k* : value of k in kmer (length of DNA subsequence).
        - *-tfidf* : the tf-idf score cutoff. Range [0.00 – 1.00]. This amount of kmers with higher tf-idf will be taken from each sequence.

    - --*useHdf* : whether to use persistent hdf to store part of the data in disk. When k is high and memory is limited, this can be useful but it is also slower than direct in-memory calculations. This is mostly needed for TF-IDF calculations on large k.
  - ***Output***:
    - *vocabCountIndex.hdf* : PyTables file containing counts of the kmers
    - *vocabularyPathCodes.hdf* : Pytables file containing Huffman codes and edge id's of the binary tree for a kmer.
    - *vocabHash*: Python pickled dictionary with kmer subsequence as key and kmer id as value. This file will be used by many other Python scripts in this project.

Step 2: Train the kmer2vec

- **kmer2vec_train.py**
  - ***Dependency***: PyTorch for neural network. Link: https://pytorch.org/
  - ***Arguments***:
    - -*vocabFilePrefix*: the output directory of *vocabulary.py* (in previous step). Please do not include any file name here, just the directory.
    - -*vocabHashFile*: the python pickled dictionary with kmer and id's from vocabulary.py script. Do not include the directory, we have already added that in previous argument.
    - - *vocabCountIndexFile*: The PyTables file containing kmer counts from vocabulary.py script. Do not include the directory, only the file name.
    - -*vocabularyPathCodesFile*: The PyTables file containing the Huffman codes of the kmers.
    - --*saveModel*: should we save the trained model in each iteration? Boolean argument and default value is false.
    - -*savedModelPath*: Provide the Path where model state will be saved.
    - --loadModel: should we load model from already existing file? Boolean argument and default value is false.
    - -*loadModelPath*: Path from where the model will be loaded.
    - -*embedding_dim*: Integer value. The embedding dimension of the kmers.
    - -*min_context_window*: Integer value. The minimum context window for a target kmer.
    - -*max_context_window*: Integer value. The maximum context window for a target kmer.
    - -*learning_rate*: Float value. The learning rate of the optimizer.
    - -*max_iteration*: Integer value. The maximum iteration for the kmer embedding learner.

        - -*kmer_per_sequence*: Integer value. How many kmers will be extracted from each sequence. Works as a kmer sampler from each sdequence.
        - -*batch_size*: Integer value. The batch size for the training.
        - -*device*: which device to use? cpu (default) / cuda:0 / cuda:1 / cuda:2 / cuda:3
        -
    - ***Output***:
        - *States_k_d.pytorch*: A pytorch file that contains the kmer embedding vectors.
        -

Instance representation, bag representation and classification

Step 3: Instance representation using Mini-batch KMeans Clustering

- **kmeans_clustering.py**
    - ***Dependency***: scikit-learn for the mini-batch implementation.
    - ***Arguments:***
        - -*vocabHashFile*: The file generated by the *vocabulary.py* script. This is a python "pickled" dictionary with kmers as key and unique id as the value. Provide the full path including the filename.
        - -*trainedModelFile*: provide the PyTorch file we created as part of the kmer2vec process. This contains the learned kmer vectors. Provide the full path including the file name.
        - -data_directory : Provide the directory of the data.
        - -kmeans : Integer value. the number of clusters in kmeans.
    - ***Output***:
        - *One ".npy" file per sample*: A numpy file that contains the instances (cluster centroids) of the sample. Example: for "sample1.FASTA" we will have "sample1_centroids.npy" in the same directory as the actual data.

Step 4: Bag representation and Classification

- **cnn_train.py**
    - ***Arguments:***
        - -*vocabHashFile*: The file generated by the *vocabulary.py* script. This is a python "pickled" dictionary with kmers as key and unique id as the value. Provide the full path including the filename.
        - -*trainedModelFile*: provide the PyTorch file we created as part of the kmer2vec process. This contains the learned kmer vectors. Provide the full path including the file name.
        - - train_dir: The directory path containing training samples.

- -*test_dir*: the directory path containing test samples.
- - *class_info*: the file containing sample names with their class labels. There should be a text file recording the class labels of all the samples. Each new line in the text file should contain a sample filename and its class-label (0/1) separated by a tab.
- - *learning_rate*: the learning rate of the deep CNN model.
- - *batch_size*: the batch size of the deep CNN model.
- - *epoch*: the total number of epoch in the deep CNN model.
- - *trial*: the number of times the experiment will be repeated.
- - *repeats*: the number of repetitions in training data augmentation process.
- - *device*: which device to use? cpu (default) / cuda:0 / cuda:1 / cuda:2 / cuda:3

Contributor:
Mohammad Arifur Rahman
PhD Student, Computer Science
George Mason University, VA, US
Email: mrahma23@gmu.edu