```
#!/usr/bin/env python

'''NER Preprocessing
'''

import numpy as np
import h5py
import argparse
import sys
import re
import codecs

# Preprocessing, feature construction, and GloVe setup

FILE_PATHS = {'CONLL': ('data/train.num.txt',
          'data/dev.num.txt',
          'data/test.num.txt',
          'data/tags.txt')}
args = {}
tag_to_idx = {}
word_to_idx = {}
longest = {1: 0}

START = '<s>'
STOP = '</s>'
START_TAG = '<t>'
STOP_TAG = '</t>'
UNKNOWN = '<unk>'

def build_sentences(file_list):
  input_s = {}
  output_s = {}
  for filename in file_list:
    if filename:
      input_s[filename] = []
      output_s[filename] = []
      with codecs.open(filename, 'r', encoding='latin-1') as f:
        print('Building sentences from ' + filename + '...')

        sentence = [word_to_idx[START]]
        tags = [tag_to_idx[START_TAG]]
        for line in f:
          line = line.split()
          if len(line) == 0: # EOS
            # Add closing word and tag
            sentence.append(word_to_idx[STOP])
            tags.append(tag_to_idx[STOP_TAG])
```

```
            input_s[filename].append(sentence)
            output_s[filename].append(tags)
            if len(sentence) > longest[1]: longest[1] = len(sentence)

            sentence = [word_to_idx[START]]
            tags = [tag_to_idx[START_TAG]]
            continue

          # If we got here, we have a valid word in the middle of a sentence
          word = word_to_idx[str(line[2])]
          sentence.append(word)
          if filename != 'data/test.num.txt':
            tag = tag_to_idx[str(line[3])]
            tags.append(tag)

        # Standardize sentence/tag length with padding
        for i in range(len(input_s[filename])):
          input_s[filename][i] = input_s[filename][i] + [word_to_idx[STOP]] \
            * (longest[1] - len(input_s[filename][i]))
          output_s[filename][i] = output_s[filename][i] + [tag_to_idx[STOP_TAG]] \
            * (longest[1] - len(output_s[filename][i]))
  return input_s, output_s

def sentences_to_windows(xs, ys, dwin):
  input_w = []
  input_t = []
  output = []
  input_w_s = []
  input_t_s = []
  output_s = []
  for i in range(len(xs)):
    padding = dwin - 1 # Each sentence already has 1 start token
    x = [word_to_idx[START]] * padding + xs[i][0:xs[i].index(word_to_idx[STOP]) + 1]
    y = [tag_to_idx[START_TAG]] * padding + ys[i][0:ys[i].index(tag_to_idx[STOP_TAG])
+ 1]

    w_s = []
    t_s = []
    o_s = []
    for j in range(dwin, len(x)):
      # x = dwin prev tags (excl current), dwin words (incl current)
      # y = current tag
      w_window = x[j-dwin+1:j+1]
      prev_tags = y[j-dwin:j]
      input_w.append(w_window)
      input_t.append(prev_tags)
```

```
      output.append(y[j])
      w_s.append(w_window)
      t_s.append(prev_tags)
      o_s.append([y[j]])

    # Standardize length with padding
    w_s = w_s + [[word_to_idx[STOP]]] * (longest[1] - len(w_s))
    t_s = t_s + [[tag_to_idx[STOP_TAG]]] * (longest[1] - len(t_s))
    o_s = o_s + [[tag_to_idx[STOP_TAG]]] * (longest[1] - len(o_s))

    # Format into sentence chunks
    input_w_s.append(w_s)
    input_t_s.append(t_s)
    output_s.append(o_s)
    w_s = []
    t_s = []
    o_s = []
  return input_w, input_t, output, input_w_s, input_t_s, output_s

def test_sentences_to_windows(xs, dwin):
  input_w_s = []

  for i in range(len(xs)):
    padding = dwin - 1
    x = [word_to_idx[START]] * padding + xs[i][0:xs[i].index(word_to_idx[STOP]) + 1]
    w_s = []
    for j in range(dwin, len(x)):
      w_window = x[j-dwin+1:j+1]
      w_s.append(w_window)

    # Standardize length with padding
    w_s = w_s + [[word_to_idx[STOP]]] * (longest[1] - len(w_s))

    # Format into sentence chunks
    input_w_s.append(w_s)
    w_s = []
  return input_w_s

def build_tag_dict(filename):
  idx = -1
  with codecs.open(filename, 'r', encoding='latin-1') as f:
    for line in f:
      info = line.split()
      abbrev = str(info[0])
      idx = int(info[1])
      tag_to_idx[abbrev] = idx
  tag_to_idx[START_TAG] = idx + 1
```

```
  tag_to_idx[STOP_TAG] = idx + 2

def build_word_dict(file_list):
  last_idx = 3
  word_to_idx[START] = 1
  word_to_idx[STOP] = 2
  for filename in file_list:
    if filename:
      with codecs.open(filename, 'r', encoding='latin-1') as f:
        for line in f:
          line = line.split()
          if len(line) == 0: continue

          word = str(line[2])
          if word not in word_to_idx:
            word_to_idx[word] = last_idx
            last_idx = last_idx + 1
  print('Built word dict with ' + str(last_idx - 1) + ' entries.')

def build_vector(filename):
  vector_dict = {}

  with codecs.open(filename, 'r', encoding='latin-1') as f:
    for line in f:
      try:
        info = line.split()
        word = info[0]
        vec = [float(x) for x in info[1:]]
      except:
        continue
      vector_dict[word] = vec
  return vector_dict

def main(arguments):
  global args
  parser = argparse.ArgumentParser(
    description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter)
  parser.add_argument('dataset', help='Data set',
          type=str)
  args = parser.parse_args(arguments)
  dataset = args.dataset
  train, valid, test, tag_dict = FILE_PATHS[dataset]
  sets = [train, valid, test]

  build_word_dict(sets)
  build_tag_dict(tag_dict)
```

```
V = len(word_to_idx)
C = len(tag_to_idx)

input_dict, output_dict = build_sentences(sets)
train_input = np.array(input_dict[train], dtype=np.int32)
train_output = np.array(output_dict[train], dtype=np.int32)
valid_input = np.array(input_dict[valid], dtype=np.int32)
valid_output = np.array(output_dict[valid], dtype=np.int32)
test_input = np.array(input_dict[test], dtype=np.int32)

dwin = 1
train_input_w, train_input_t, train_output_memm, train_input_w_s, \
  train_input_t_s, train_output_memm_s, = sentences_to_windows(input_dict[train], \
  output_dict[train], dwin)
valid_input_w, valid_input_t, valid_output_memm, valid_input_w_s, \
  valid_input_t_s, valid_output_memm_s = sentences_to_windows(input_dict[valid], \
  output_dict[valid], dwin)
test_input_w_s = test_sentences_to_windows(input_dict[test], dwin)

train_input_w = np.array(train_input_w, dtype=np.int32)
train_input_t = np.array(train_input_t, dtype=np.int32)
train_output_memm = np.array(train_output_memm, dtype=np.int32)

valid_input_w = np.array(valid_input_w, dtype=np.int32)
valid_input_t = np.array(valid_input_t, dtype=np.int32)
valid_output_memm = np.array(valid_output_memm, dtype=np.int32)
valid_input_w_s = np.array(valid_input_w_s, dtype=np.int32)
valid_input_t_s = np.array(valid_input_t_s, dtype=np.int32)
valid_output_memm_s = np.array(valid_output_memm_s, dtype=np.int32)

filename = args.dataset + '.hdf5'
with h5py.File(filename, 'w') as f:
  f['train_input'] = train_input
  f['train_output'] = train_output
  f['valid_input'] = valid_input
  f['valid_output'] = valid_output
  f['test_input'] = test_input

  f['train_input_w'] = train_input_w
  f['train_input_t'] = train_input_t
  f['train_output_memm'] = train_output_memm
  f['train_input_w_s'] = train_input_w_s
  f['train_input_t_s'] = train_input_t_s
  f['train_output_memm_s'] = train_output_memm_s


  f['valid_input_w'] = valid_input_w
```

```
    f['valid_input_t'] = valid_input_t
    f['valid_output_memm'] = valid_output_memm
    f['valid_input_w_s'] = valid_input_w_s
    f['valid_input_t_s'] = valid_input_t_s
    f['valid_output_memm_s'] = valid_output_memm_s

    f['test_input_w_s'] = test_input_w_s

    f['nfeatures'] = np.array([dwin * 2], dtype=np.int32)
    f['nwords'] = np.array([V], dtype=np.int32)
    f['nclasses'] = np.array([C], dtype=np.int32)


if __name__ == '__main__':
  sys.exit(main(sys.argv[1:]))
```