Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Assignment

Write a python program that parses all XML files in the folder given as argument and recognizes and classifies drug names. The program must use a neural network approach.

```
$ python3 ./ml-NER.py data/Devel/
DDI-DrugBank.d278.s0|0-9|Enoxaparin|drug
DDI-DrugBank.d278.s0|93-108|pharmacokinetics|group
DDI-DrugBank.d278.s0|113-124|eptifibatide|drug
DDI-MedLine.d88.s0|15-30|chlordiazepoxide|drug
DDI-MedLine.d88.s0|33-43|amphetamine|drug
DDI-MedLine.d88.s0|49-55|cocaine|drug
DDI-MedLine.d88.s1|82-95|benzodiazepine|drug
...
```

Outline

General Structure

The general structure is basically the same than for the traditional ML approach:

- Two programs: one learner and one classifier.
- The learner loads the training (Train) and validation (Devel) data, formats/encodes it appropiately, and feeds the model with the data plus its ground truth.
- The classifier loads the test data, formats/encodes it in the same way that was used in training, and feeds it to the model to get a prediction.

In the case of NN, we don't need to extract features (though we do need some encoding)

Outline

Outline

Learner - Main program

```
def learn(traindir, validationdir, modelname) :
   '''
   learns a NN model using traindir as training data, and validationdir
   as validation data. Saves learnt model in a file named modelname
   '''
   # load train and validation data in a suitable form
   traindata = load_data(traindir)
   valdata = load_data(validationdir)

   # create indexes from training data
   max_len = 100
   idx = create_indexs(traindata, max_len)

   # build network
   model = build_network(idx)

   # encode datasets
   Xtrain = encode_words(traindata, idx)
   Ytrain = encode_tags(traindata, idx)
   Xval = encode_words(valdata, idx)
   Yval = encode_tags(valdata, idx)

   # train model
   model.fit(Xtrain, Ytrain, validation_data=(Xval,Yval))

   # save model and indexs, for later use in prediction
   save_model_and_indexs(model, idx, modelname)
```

Outline

Classifier - Main program

```
def predict(modelname, datadir, outfile) :
   '''
   Loads a NN model from file 'modelname' and uses it to extract drugs
   in datadir. Saves results to 'outfile' in the appropriate format.
   '''

   # load model and associated encoding data
   model, idx = load_model_and_indexs(modelname)
   # load data to annotate
   testdata = load_data(datadir)

   # encode dataset
   X = encode_words(testdata, idx)

   # tag sentences in dataset
   Y = model.predict(X)
   # get most likely tag for each word
   Y = [[idx['tags'][np.argmax(y)] for y in s] for s in Y]

   # extract entities and dump them to output file
   output_entities(testdata, Y, outfile)

   # evaluate using official evaluator.
   evaluation(datadir,outfile)
```

Outline

Required functions - load_data

```
def load_data(datadir)
```

- Used by: Learner, Classifier
- Input: Receives a directory containing XML files.
- Output: Parses XML files in given directory, tokenizes each sentence, extracts ground truth BIO tags for each token, and returns the dataset as a dictionary. Dictionary keys are the sentence id, and values are the list of token tuples (word, start, end, ground truth).
- Example:

```
>>> load_data('data/Train')
{'DDI-DrugBank.d370.s0':  [('as', 0, 1, 'O'), ('differin', 3, 10, 'B-brand'),
                           ('gel', 12, 14, 'O'), ..., ('with', 343, 346, 'O'),
                           ('caution', 348, 354, 'O'), ('.', 355, 355, 'O')],
 'DDI-DrugBank.d370.s1':  [('particular', 0, 9, 'O'), ('caution', 11, 17, 'O'),
                           ('should', 19, 24, 'O'), ...,('differin', 130, 137, 'B-brand'),
                           ('gel', 139, 141, 'O'), ('.', 142, 142, 'O')],
 ...
}
```

Use XML parsing and tokenization functions from previous exercises

Required functions - create_indexs

```
def create_indexs(datadir, max_length)
```

- Used by: Learner
- Input: Receives a dataset produced by `load_data`, and the maximum length in a sentence
- Output: Creates a set of words seen in the data and a set of BIO tags. Enumerates those sets, assigning a unique integer to each element. Returns these mappings in a single dictionary, with an additional entry for the given max_length value.
- Example:

```
>>> create_indexs(traindata)
{'words': {'<PAD>':0, '<UNK>':1, '11-day':2, 'murine':3, 'criteria':4,
          'stroke':5, ... 'carbidopa-levodopa':8510, 'generation':8511,
          'terfenadine*': 8512 }
 'tags': {'<PAD>':0, 'B-group':1, 'B-drug_n':2, 'I-drug_n':3, 'O':4,
          'I-group':5, 'B-drug':6, 'I-drug':7, 'B-brand':8, 'I-brand':9}
 'maxlen' : 100 }
```

Add a <PAD> code to both 'words' and 'tags' indexes, with value 0. Add also an <UNK> code to 'words' with value 1. The coding of the rest of the words or tags is arbitrary.

Required functions - build_network

```
def build_network(idx) :
   '''
   Used by: Learner
   Input: Receives the index dictionary with the encondings of words and
     tags, and the maximum length of sentences.
   Output: Returns a compiled Keras neural network
   '''
   # sizes
   n_words = len(idx['words'])
   n_tags = len(idx['tags'])
   max_len = idx['maxlen']

   # create network layers
   inp = Input(shape=(max_len,))
   ## ... add missing layers here ... #
   out = # final output layer

   # create and compile model
   model = Model(inp,out)
   model.compile()  # set appropriate parameters (optimizer, loss, etc)

   return model
```

Required functions - build_network

```
def build_network(idx) :
```

- LSTMs are useful for sequence tagging tasks such as NER.
- You will need to add one Embedding layer after the input, that is where the created indexes will become handy.
- You can base your model in these examples: [1],[2],[3],[4],[5],[6]
  Note: some instructions may require to be adapted, depending on your Keras version.
  Note: you don't need to follow the **whole** example, only the network construction part.

Required functions - encode_words

```
def encode_words(dataset, idx) :
```

- Used by: Learner, Classifier
- Input: Receives a dataset produced by `load_data`, and the index dictionary produced by `create_indexs`
- Output: Returns the dataset as a list of sentences. Each sentence is a list of integers, corresponding to the code of each **word** in the sentence. If the word is not in the index, the code for <UNK> is used. If the sentence is shorter than `max_len` it is padded with the code for <PAD>.
- Example:

```
>>> encode_words(traindata,idx)
[ [6882 1049 4911 ...  0 0 0]
  [2290 7548 8069 ...  0 0 0]
  ...
  [5964 5183 3519 ...  0 0 0]
  [2002 6582 7518 ...  0 0 0] ]
```

Required functions - encode_tags

```
def encode_tags(dataset, idx) :
```

- Used by: Learner
- Input: Receives a dataset produced by `load_data`, and the index dictionary produced by `create_indexs`
- Output: Returns the dataset as a list of sentences. Each sentence is a list of integers, corresponding to the code of the **BIO tag** for each word. If the sentence is shorter than `max_len` it is padded with the code for `<PAD>`.
- Example:

```
>>> encode_tags(traindata,idx)
[ [ [6] [9] [6] ... [0] [0] [0] ]
  [ [6] [6] [6] ... [0] [0] [0]]
  ...
  [ [6] [8] [6] ... [0] [0] [0] ]
  [ [6] [6] [6] ... [0] [0] [0] ] ]
```

Note: The shape of the produced list may need to be adjusted depending on the architecture of your network and the kind of output layer you use.

Required functions - Model saving and loading

```
def save_model_and_indexs(model, idx, filename) :
```

- Used by: Learner
- Input: Receives a trained model, an index dictionary, and a string.
- Output: Stores the model in a file named `filename.nn`, and the indexs in a file named `filename.idx`

```
def load_model_and_indexs(filename) :
```

- Used by: Classifier
- Input: Loads a model from `filename.nn`, and the indexs from `filename.idx`.
- Output: Returns the loaded model and indexs

Note: Use Keras `model.save` and `keras.models.load_model` functions to save/load the model.

Note: Use your preferred method (pickle, plain text, etc) to save/load the index dictionary.

Required functions - output_entities

```
def output_entities(dataset, preds, outfilename)
```

- Used by: Classifier
- Input: Receives a dataset produced by `load_data`, and the corresponding tags predicted by the model.
- Output: Prints the detected entities in file *outfilename* in the appropriate format for the evaluator: one line per entity, fields separated by '|', field order: *id*, *offset*, *name*, *type*.
- Example:

```
>>> output_entities(dataset, preds, filename)
    DDI-DrugBank.d283.s4|14-35|bile acid sequestrants|group
    DDI-DrugBank.d283.s4|99-104|tricor|group
    DDI-DrugBank.d283.s5|22-33|cyclosporine|drug
    DDI-DrugBank.d283.s5|196-208|fibrate drugs|group
    DDI-DrugBank.d283.s4|14-35|bile acid sequestrants|group
    DDI-DrugBank.d283.s5|220-225|tricor|group
    ...
```

Note: Most of this function can be reused from NER-ML exercise.

Required functions - evaluation

```
def evaluation(datadir, outfile)
```

- Used by: Classifier
- Input: Receives a directory with ground truth data, and a file with entities extracted by the model
- Output: Runs the official evaluator and gets the results

Note: Reuse this function from previous exercises

Outline

Build a good NN-based drug NERC

Strategy: Experiment with different NN architectures and possibilities.
Some elements you can play with:

- Embedding dimension
- Number of LSTM units
- Used optimizer
- Number and kind of layers
- Adding a CRF layer after the LSTM
- Using lowercased and/or non lowercased word embeddings
- Initialitzing embeddings with available pretrained model
- Using extra input (e.g. suffix embeddings, prefix embeddings, PoS embbedings, ...)
- ...

Build a good NN-based drug NERC

Warnings:

- Neural Network training uses randomization, so different runs of the same program will produce different results. For repeatable results, use a random seed.
- During training, Keras reports accuracy on training set and on validation set. Those values are usually over 90%. However, this is due to the fact that most of the words have tag "O" (non-drug). 90% accuracies correspond to F_1 values around 25%. To get a reasonable F_1, accuracy must reach about 97%. To precisely evaluate how your model is doing, do not rely on reported accuracy: run the classifier on the Development set and use the evaluator.

Exercise Goals

Goal 5:
Get an overall F_1 score of at least 0.70 on **Devel** dataset. (65% is acceptable for a lower grade)

Deliverables

Write one report (max about 5 pages) describing:

- Used architecture
- Performed experiments, tried/discarded/selected options.

The report must include:

- Code for the `build_network` function
- Output of the evaluator on **Devel** and **Test** datasets.

The report must be a PDF file, or a Jupyter notebook (no need that it is are executable, use it only as a presentation support)