Master in Artificial Intelligence

Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Session 6 - DDI using neural networks

Assignment

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

```
$ python3 ./nn-DDI.py data/Devel/
DDI-DrugBank.d278.s0|DDI-DrugBank.d278.s0.e0|DDI-DrugBank.d278.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e2|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e1|DDI-MedLine.d88.s0.e2|0|null
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e1|1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e2|1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e2|DDI-DrugBank.d398.s0.e3|0|null
DDI-DrugBank.d398.s1|DDI-DrugBank.d398.s1.e0|DDI-DrugBank.d398.s1.e1|0|null
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e0|DDI-DrugBank.d211.s2.e5|1|mechanism
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e1|DDI-DrugBank.d211.s2.e2|0|null
...
```

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


```
1  def learn(traindir, validationdir, modelname) :
2      '''
3      learns a NN model using traindir as training data, and validationdir
4      as validation data. Saves learnt model in a file named modelname
5      '''
6      # load train and validation data in a suitable form
7      traindata = load_data(traindir)
8      valdata = load_data(validationdir)
9
10     # create indexes from training data
11     max_len = 100
12     idx = create_indexs(traindata, max_len)
13
14     # build network
15     model = build_network(idx)
16
17     # encode datasets
18     Xtrain = encode_words(traindata, idx)
19     Ytrain = encode_tags(traindata, idx)
20     Xval = encode_words(valdata, idx)
21     Yval = encode_tags(valdata, idx)
22
23     # train model
24     model.fit(Xtrain, Ytrain, validation_data=(Xval,Yval))
25
26     # save model and indexs, for later use in prediction
27     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 pair
    Y = [[idx['tags'][np.argmax(y)] for y in Y]

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

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

Outline

Required functions - `load_data`


```
1  def load_data(datadir)
```

- Used by: Learner, Classifier
- Input: Receives a directory containing XML files.
- Output: Parses XML files in given directory, tokenizes/analyzes each sentence, extracts ground truth class for each example, and returns the dataset as a list of examples. Each example corresponds to a drug pair in a sentence, and contains: sentence id, entity1 id, entity2 id, ground truth class, and a list of sentence tokens (each token containing any needed information: word, lemma, PoS, offsets, etc)

Use XML parsing and tokenization functions from previous exercises. Adding a PoS tagger or lemmatizer may be useful but it is not a strict requirement.

Masking the target drugs as e.g. `<DRUG1>`, `<DRUG2>`, and `<DRUG_OTHER>` will help the algorithm generalize and avoid it focusing in the drug names, which are not relevant for the task.

Required functions - load_data (cont.)

- Example:

```
>>> load_data('data/Train')
[ ['DDI-DrugBank.d66.s0', 'DDI-DrugBank.d66.s0.e0', 'DDI-DrugBank.d66.s0.e1',
   'null', [('<DRUG1>', '<DRUG1>', '<DRUG1>'), ('-', '-', ':'),
            ('Concomitant', 'concomitant', 'JJ'), ('use', 'use', 'NN'),
            ('of', 'of', 'IN'), ('<DRUG2>', '<DRUG2>', '<DRUG2>'),
            ('and', 'and', 'CC'), ('<DRUG_OTHER>', '<DRUG_OTHER>', '<DRUG_OTHER>'),
            ('may', 'may', 'MD'),
            ...
            ('bowel', 'bowel', 'NN'), ('syndrome', 'syndrome', 'NN'), ('.', '.', '.')
           ] ]
...
  ['DDI-MedLine.d94.s12', 'DDI-MedLine.d94.s12.e1', 'DDI-MedLine.d94.s12.e2',
   'effect', [('The', 'the', 'DT'), ('uptake', 'uptake', 'NN'),
            ('inhibitors', 'inhibitor', 'NNS'), ('<DRUG_OTHER>', '<DRUG_OTHER>',
            '<DRUG_OTHER>'), ('and', 'and', 'CC'), ('<DRUG1>', '<DRUG1>', '<DRUG1>'),
            ...
            ('potentiated', 'potentiate', 'VBD'), ('the', 'the', 'DT'),
            ('positive', 'positive', 'JJ'), ('inotropic', 'inotropic', 'JJ'),
            ('effects', 'effect', 'NNS'), ('of', 'of', 'IN'),
            ('<DRUG2>', '<DRUG2>', '<DRUG2>'), ('in', 'in', 'IN'),
            ...  ] ]
]
```

Required functions - create_indexs


```
1  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 interaction classes. 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':  {'null':0, 'mechanism':1, 'advise':2, 'effect':3, 'int':4 }
 'maxlen' :  100 }
```

Add a <PAD> code to 'words' index, 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. You may add entries with indexes for other element you may want to use (lemmas, PoS, etc)

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`

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

- DDI is not sequence tagging task (which assign one label per word), but a sentence classification, where a single label is assigned to the whole sentence (or sentence+pair in this case).
- The problem may be approached with an LSTM, but since it produces a label per word, some layers need to be added to convert the output to a single class. A good alternative is using a CNN, which also produce good results for text processing, and are more straightforward to apply to this kind of tasks
- 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],
  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

```
1  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>. You may adapt this function to return more than one list if you want to use different inputs (e.g. words, lemma, PoS, ...)
- Example:

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

Required functions - encode_tags


```
1   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 class indexes, one per sentence.
- Example:

```
>>> encode_tags(traindata,idx)
[ [0] [0] [2] ...  [4] [0] [0] [1] [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


```
1  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`

```
1  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_interactions


```
1  def output_interactions(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: *sentence_id*, *e1_id*,, *e2_id*, *ddi*, *type*.
- Example:

```
>>> output_interactions(dataset, preds, filename)
DDI-DrugBank.d278.s0|DDI-DrugBank.d278.s0.e0|DDI-DrugBank.d278.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e1|0|null
DDI-MedLine.d88.s0|DDI-MedLine.d88.s0.e0|DDI-MedLine.d88.s0.e2|0|null
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e1|1|effect
DDI-DrugBank.d398.s0|DDI-DrugBank.d398.s0.e0|DDI-DrugBank.d398.s0.e2|1|effect
DDI-DrugBank.d398.s1|DDI-DrugBank.d398.s1.e0|DDI-DrugBank.d398.s1.e1|0|null
DDI-DrugBank.d211.s2|DDI-DrugBank.d211.s2.e0|DDI-DrugBank.d211.s2.e5|1|mechanism
...
```

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

Required functions - `evaluation`


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

- Used by: Classifier
- Input: Receives a directory with ground truth data, and a file with interactions 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 DDI detector

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

- Embedding dimension
- Using LSTM+CNN or just CNN
- Used optimizer
- Number and kind of layers
- Using lowercased and/or non lowercased word embeddings
- Initialitzing embeddings with available pretrained model
- Using extra input (e.g. lemma embeddings, PoS embeddings, suffix/prefix embbedings, ...)
- Using pretrained transformers such as Bert as the first layers of your network.
- ...

Build a good NN-based DDI detector

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 82%. However, this is due to the fact that most of the pairs have interaction "`null`" (no-interaction). 82% accuracies correspond very low F_1 values. To get a reasonable F_1, accuracy must reach at least 88%. 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 0.60 on **Devel** dataset.

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)