# Re-structuring

The part that retrieves data from WordNet and stores them is OK, and does not need to be changed.

Regarding the vocabulary, reading data from a SLC corpus, et cetera there is a list of modifications that should have been implemented long ago:

* Make 2 separate folders, for “InputData\_sense-labeled” and “InputData\_standardText”
* The sense-labeled corpus must be *rewritten, lowercased.* There should be no difference in processing.
* I must write a protocol for experiments, that includes: checking what the Neural Network reads, checking that we can overfit on a mini-fragment
* New guideline: *no more manual switches*. We should have a folder with mini-corpuses for mini-experiments (both normal text and sense-labeled).

## Step by step

### CreateGraphInput.py

Functions:

1. reset()
2. reset\_embeddings()
3. exe\_from\_input\_to\_vectors(do\_reset, compute\_single\_prototype, sp\_method, vocabulary\_from\_senselabeled)

We must change the vocabulary\_from\_senselabeled parameter into a senselabeled\_or\_text parameter, that decides which folder we use, etc.

Moreover, we must have reset(senselabeled\_or\_text) and reset\_embeddings(senselabeled\_or\_text)

We also change the location of the graph and grapharea\_matrix: into Graph/DatasetTypeSubfolder

### senselabeled\_or\_text = True

**SLC.organize\_splits()**

Changed into:

SLC.organize\_splits(xml\_fnames)

with xml\_fnames = [**'semcor.xml'**]

The existing code works as intended, no need to modify. We recall that:

train\_root, valid\_root, test\_root = organize\_subcorpus(xml\_fpath, 0.8)  
write\_splits\_subcorpus(xml\_fpath, train\_root, valid\_root, test\_root)

### Vocabulary

It is now opportune for us to change:

vocabulary = **V.get\_vocabulary\_df**(senselabeled\_or\_text=vocabulary\_from\_senselabeled,  
 corpus\_txt\_fpaths=[vocab\_text\_fpath],  
 out\_vocabulary\_h5\_filepath=outvocab\_filepath, min\_count=2, lowercase=**True**)

**if** os.path.exists(out\_vocabulary\_h5\_filepath):  
 vocab\_df = pd.read\_hdf(out\_vocabulary\_h5\_filepath, mode=**'r'**)  
 logging.info(**"\*\*\* The vocabulary was loaded from the file "** + out\_vocabulary\_h5\_filepath)

We should not get the vocabulary\_of\_globals.h5 file only in 1 place, but in Vocabulary/SenseLabeled or Vocabulary/StandardText

**if** senselabeled\_or\_text:  
 vocabulary\_wordfreq\_dict = build\_vocabulary\_dict\_from\_senselabeled()  
**else**:  
 vocabulary\_wordfreq\_dict = build\_vocabulary\_dict\_fromtext(corpus\_txt\_fpaths)

**build\_vocabulary\_dict\_from\_senselabeled(lowercase)**:

vocab\_dict = {}  
  
tokens\_toexclude = [Utils.EOS\_TOKEN] *# + list(string.punctuation)  
# Commas and punctuation signs are present in the Sense-Labeled Corpus as separate tokens.  
# Therefore, it makes sense to keep them in the vocabulary, and thus in the graph, as globals*slc\_split\_names = [Utils.TRAINING, Utils.VALIDATION] *# , , Utils.TEST*

Consideration: since SemCor is smaller, it may make sense to create the vocabulary not only from the training set, but also from the validation set.

Not an extremely heavy choice anyway, given the 80-10-10 split. Let’s just keep T+V

**for** token\_dict **in** SLC.read\_split(slc\_split\_name):  
 token = VocabUtils.process\_word\_token(token\_dict, lowercase)  
 **if** token **not in** tokens\_toexclude:  
 **try**:  
 prev\_freq = vocab\_dict[token]  
 vocab\_dict[token] = prev\_freq + 1

Creation of the vocabulary with process\_word\_token(td, lowercase) has worked correctly so far.

However, this is the insertion point for the *lemmatized* forms of words in the vocabulary. This was brought about by, for instance, having ‘eyelids’ but not ‘eyelid’ in SemCor, where the second form actually has WordNet senses & data.

How to handle this?

Since the frequency of a word is not relevant in the vocabulary apart from whether it is < min\_count or not, I can just add +1 to the frequency of the lemmatized word as well.

We have to take the lemmatizer for this, from:

lemmatizer = nltk.stem.WordNetLemmatizer()  
lemmatized\_forms = [LN.lemmatize\_term(word, lemmatizer) **for** word **in** vocab\_wordls]

**build\_vocabulary\_dict\_fromtext(corpus\_txt\_fpaths)**:

It doesn’t need to be changed, since it gives us 33,278 – correctly – on WikiText-2.

### Compute Embeddings

reset\_embeddings(senselabeled\_or\_text). #ok

single\_prototypes\_file #should go into the appropriate subfolder

**compute\_single\_prototype\_embeddings**(vocabulary\_df, spvs\_out\_fpath, method):

**if** method == Method.DISTILBERT:

… # currently not implemented

**else**: *# i.e. elif method == Method\_for\_SPV.FASTTEXT:*

fasttext\_vectors = EFT.load\_fasttext\_vectors()

**for** idx\_word\_freq\_tpl **in** vocabulary\_df.itertuples():

word = idx\_word\_freq\_tpl[1]

word\_vector = fasttext\_vectors[word]

word\_vectors\_lls.append(word\_vector)

embds\_nparray = np.array(word\_vectors\_lls)  
 np.save(spvs\_out\_fpath, embds\_nparray)

### WordNetData and Post-processing

vocabulary\_ls = RID.retrieve\_data\_WordNet(vocabulary)

we must adapt to the subfolders in InputData and Vocabulary

**PI.prepare**(vocabulary\_ls, sp\_method)

*# Phase 1 - Preprocessing: eliminating quasi-duplicate definitions and examples, and lemmatizing synonyms & antonyms*preprocess(vocabulary\_ls, inputdata\_folder)

*# Phase 2 - Create the Vocabulary table with the correspondences (wordSense, integer index).*create\_senses\_indices\_table(inputdata\_folder, vocabulary\_folder):

* Possible insertion point: until now, we add the dummySense if the word is one of those without senses. However, it is opportune to add it only if the *lemmatized form* has no senses. (remember to exclude words with <3chars, like ‘its’)
* It is also necessary to send the senselabeled\_or\_text specification, so we know which filepaths to use (where to find the vocabulary, etc.)

vocab\_fpath, vocabulary\_df, vocabulary\_words\_ls

and also input\_folder\_fpath.

defs\_input\_filepath, examples\_input\_filepath, defs\_input\_db, examples\_input\_db.

output\_filepath.

out\_indicesTable\_db = sqlite3.connect(output\_filepath)

out\_indicesTable\_db\_c = out\_indicesTable\_db.cursor()

out\_indicesTable\_db\_c.execute(**'''CREATE TABLE IF NOT EXISTS …**

**word\_senses\_series\_from\_defs** =   
 defs\_input\_db[Utils.DEFINITIONS][Utils.SENSE\_WN\_ID]

**word\_senses\_ls** = [sense\_str **for** sense\_str **in** word\_senses\_series\_from\_defs **if** Utils.get\_word\_from\_sense(sense\_str) **in** vocabulary\_words\_ls]

**for** wn\_id **in** word\_senses\_ls:

…out\_indicesTable\_db\_c.execute(

**"INSERT INTO indices\_table VALUES (?,?,?,?,?,?)"**,

(wn\_id, my\_vocabulary\_index,start\_defs\_count,   
 end\_defs\_count, start\_examples\_count, end\_examples\_count))

*# add the word to the set of words that do have a sense*words\_with\_senses\_set.add(Utils.get\_word\_from\_sense(wn\_id))

words\_without\_senses\_set = set(vocabulary\_words\_ls).difference(words\_with\_senses\_set)

\*\*\* XYZ

Insertion point:

for all words in words\_without\_senses\_set:

if their lemmatized form (that we use for senses-processing) has a sense   
 and they are >3 characters long,   
they do not need the addition of a dummySense.

**for** word **in** words\_without\_senses\_set:  
 *# no definitions nor examples to add here. We will add the global vector as the vector of the dummy-sense.* dummy\_wn\_id = word + **'.'** + **'dummySense'** + **'.01'**  out\_indicesTable\_db\_c.execute(**"INSERT INTO indices\_table VALUES (?,?,?,?,?,?)"**, (dummy\_wn\_id, my\_vocabulary\_index,…

Then,

*# Phase 3 - get the sentence embeddings for definitions and examples, using BERT or FasText, and store them*CE.compute\_elements\_embeddings(Utils.DEFINITIONS, embeddings\_method, inputdata\_folder)  
CE.compute\_elements\_embeddings(Utils.EXAMPLES, embeddings\_method, inputdata\_folder)  
  
*# Phase 4 - use PCA dimensionality reduction for definitions and examples, for future graph processing*apply\_PCA\_to\_defs\_examples(embeddings\_method, inputdata\_folder)

### Graph