Natural Language Understanding (2016–2017)

School of Informatics, University of Edinburgh
Mirella Lapata and Frank Keller

Assignment 1: Distributional Models of Semantics

The assignment is due 17th February 2017, 16:00.

Submission Information Your solution should be delivered in two parts and uploaded to Blackboard Learn.

For your writeup:

- Write up your answers in a file titled `<EXAM NO>.pdf`. For example, if your exam number is `B123456`, your corresponding PDF should be named `B123456.pdf`.
- The answers should be clearly numbered and can contain text, diagrams, graphs, formulas, as appropriate. Do not repeat the question text.
- On Blackboard Learn, select the Turnitin Assignment "Assignment 1a: Distributional Models of Semantics ANSWERS". Upload your `<EXAM NO>.pdf` to this assignment, and use the submission title `<EXAM NO>`. So, for the above example, you should enter the submission title `B123456`.
- Please make sure you have submitted the right file. We cannot make concessions for students who turn in incomplete or incorrect files by accident.

For your code and output files:

- Compress your code files `question1.py`, `question2.py` and any output `*.txt` files we explicitly ask you to produce into a ZIP file named `<EXAM NO>.zip`. For example, if your exam number is `B123456`, your corresponding ZIP should be named `B123456.zip`.
- On Blackboard Learn, select the Turnitin Assignment "Assignment 1b: Distributional Models of Semantics CODE". Upload your `<EXAM NO>.zip` to this assignment, and use the submission title `<EXAM NO>`.

Good Scholarly Practice Please remember the University requirement as regards all assessed work for credit. Details and advice about this can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

and links from there. Note that, in particular, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

Assignment Data The necessary files for this assignment are available on the NLU course page (excluding `bnc.vert`) and on NLU's project space, which is accessible from DICE machines at `/afs/inf.ed.ac.uk/group/project/nlu/assignment1`. If you are working on your private machine, make sure you are able to access the data.

Python Virtual Environment For both assignments you will be using Python along with a few open-source packages. These packages cannot be installed directly, so you will have to create a virtual environment. We are using virtual enviroments to make the installation of packages and retention of correct versions as simple as possible. You can read here if you want to learn more about virtual environments: `https://virtualenv.pypa.io/en/stable/`.

Open a terminal on a DICE machine and follow these instructions. We are expecting you to enter these commands in one-by-one. Waiting for each command to complete will help catch any unexpected warnings and errors.

1. Change directory to home and create a virtual environment.
```
$> cd
$> virtualenv --distribute virtualenvs/nlu
```
2. Navigate to and activate the virtual environment.
```
$> cd virtualenvs/nlu
$> source ./bin/activate
```
3. Install the python packages we need. Once the correct virtual environment is activated, `pip install` will install packages to that virtual environment only. These commands will produce long outputs.
```
$> pip install -U pip
$> pip install -U setuptools
$> pip install numpy
$> pip install gensim
$> pip install pandas
```

You should now have all the required packages installed. You only need to create the virtual environment (step 1) and perform the package installations (step 3) **once**. However, make sure you activate your virtual environment (step 2) **every time** you open a new terminal to work on your NLU assignments. Remember to use the `deactivate` command to disable the virtual environment when you don't need it. If you encounter any unexpected issues, please e-mail the course TA's as soon as possible.

Question 1 [50 marks]

Carefully examine the provided `question1.py` file. Please complete the code for the functions given, as indicated. Do not use `numpy` for any vector operations. Only use `gensim` for the parts that explicitly ask you to. Give answers for the questions in your `<EXAM NO>.pdf` file.

In this assignment, we will build models for the lexical substitution task. In this task, we are given a target word in a sentence and a list of substitutes, i.e., candidate words that could be used in place of the target word. The model's job is to replace the target word with a substitute that preserves the meaning of the sentence. Table 1 gives an example of two sentences and their substitutes. Target words are shown in bold face and the correct substitute for each target is shown in italics.

Sentences	Substitutes
1. We wanted to give our client more than just a list of **bugs**.	insect, *error*, addiction
2. So I put fence **posts** all the way around the clearing.	mail, employment, *boundary*

Table 1: Lexical Substitution Sentences

A naive lexical substitution model would substitute a target word with its most similar word irrespectively of the sentence in which it appears. In example (1) in Table 1, we would compute the similarity of **bugs** with each of its substitutes (i.e., insect, error, addiction) and select the most similar one. Instead of this naive model, we will use the context-sensitive distributional models of Dinu and Lapata (2010). In particular, we will use simple vector addition, simple vector multiplication, and LDA topic models for the lexical substitution task.

In the assignment we will work with lemmatised words. The file `data/vocabulary.txt` consists of the 20,000 most frequent words in the British National Corpus (BNC). The first four lines of the file are:

```
have.v
do.v
will.v
say.v
```

The position of the word in the file indicates its index (starting from 0). The index of `have.v` is 0 as it is in the first line of the file. The indices for `do.v`, `will.v`, and `say.v` are 1, 2, and 3, respectively. The file `data/word_contexts.txt` contains a word-by-word frequency matrix. For example, the first line in the file is:

```
4999 1:69023 7:67224 4:54633 3:41610 6:31345 20:31264 2:29431 ...
```

which indicates that word index 0 (i.e., `have.v`) occurs with 4,999 context words. It co-occurs with word 1 (i.e., `do.v`) 69,023 times, with word 7 67,224 times, and so on. The context words define the dimensions of your vector space model. We currently use 5,000 context words as our base vectors (i.e., our dimensions). Each target word is represented as a vector using the context words as dimensions.

Note that `word_contexts.txt` uses sparse encoding, i.e. context words with co-occurrence frequency equal to 0 are not shown in the file. For this assignment, your implementation should be able to internally handle both sparse and full versions of these vectors.

- For sparse vectors use lists of index-frequency tuples. For example, the python representation for the above vector is:

  ```
  [ (1, 69023), (7, 67224), (4, 54633), (3, 41610), ...  ]
  ```

- For full vectors use lists of frequencies, where the value at index i equals to the frequency of the context word. The full vector for the above line is:

  ```
  [ 0, 69023, 29431, 41610, 54633, 0, 31345, 67224, ...  ]
  ```

(a) Complete the function named `load_corpus` which will load a corpus of word vectors from the files `vocabulary.txt` and `word_contexts.txt`. We will use the frequency of a context word as the magnitude of its base vector: When loading the vector of target word w, for every context word–frequency pair $c : n$, assign the frequency value n to the c-th index of the vector. We will use the term “frequency space” to refer to this vector space. You can think of each line (target word) in `word_contexts.txt` as a document. Context words of the target word correspond to the words contained in a document. The function `load_corpus` returns a list of *sparse* frequency vectors, as well as mappings from words in the vocabulary to their corresponding vector and vice-versa. [5 marks]

(b) Complete the function `cosine_similarity` to compute the cosine similarity between two vectors. Make sure that it works for both sparse and full vectors. [5 marks]

(c) What are the similarity scores between *house.n*, *home.n*, and *time.n* in the *frequency space*? Do you think these similarities reflect our intuition that *house.n* is most similar to *home.n*? [5 marks]

(d) Complete the function `tf_idf` which converts your existing vector space into tf-idf based vector space and returns the tf-idf vectors for documents. Use the following formula to calculate the tf-idf value for term *i* in document *j*:

$$tf\text{-}idf_{i,j} = (1 + log_2(tf_{i,j})) * (1 + log_2(\frac{N}{df_i}))$$

where $tf_{i,j}$ is the number of times term *i* occurs in document *j*, df_i is the number of documents that contain term *i*, and *N* is the total number of documents. Make sure that it works for both sparse and full vectors. [5 marks]

(e) What are the similarity scores between *house.n*, *home.n*, and *time.n* in the *tf-idf* space? Does your *tf-idf* space capture lexical similarity better than the *frequency* space? [5 marks]

(f) Complete the function word2vec which builds a word2vec vector model for the file `data/bnc.vert`, with dimensionality 100 and window size 5. Use the lemmas provided in the corpus. You may use the provided function `bnc_sentences` to extract training sentences from the corpus. Use the Gensim class `models.Word2Vec` to build the word2vec model.[1]

Try out different `learning rate`[2], `downsampling rate`[3], and `negative sampling`[4] parameters on a subset of the full training file (try 50,000 sentences). You can evaluate the model's accuracy on `data/accuracy_test.txt` using `models.Word2Vec`'s `accuracy` method. Which parameters give you the best results? What are some general observations? Train a model on the full set with the parameter settings you found optimal. (This may take up to 120 minutes). [5 marks]

(g) What are the similarity scores between *house-n*, *home-n*, and *time-n* in your word2vec model? Does the word2vec model capture the similarities better than the *tf-idf* space? [5 marks]

(h) Complete the function named `lda` which builds an LDA model with 100 topics using the frequency vector space (see question (a)). In LDA, each target word is a distribution over topics, and each topic a distribution over context words. Use the Gensim class `models.ldamodel.LdaModel` to build the model with 10 passes, and set the parameter `update_every=0`. (This might take up to 15 minutes). [5 marks]

[1] `https://radimrehurek.com/gensim/models/word2vec.html`

[2] From low (e.g., 0.01) to high (e.g., 0.05).

[3] Off (0.0), and from high (e.g., 0.1) to low (e.g., 0.00001).

[4] Within the range $[0 - 10]$.

(i) In LDA, vectors for target words w are represented as:

$$\mathbf{v}(\mathbf{w}) = (P(z_0|w), P(z_1|w), \ldots P(z_n|w))$$

where z_i is topic i in the LDA model (equation (1) of Dinu and Lapata (2010)). In Gensim, this vector can be retrieved by querying the model with the word's frequency vector, and is itself represented as a sparse vector.

What are the similarity scores between *house.n*, *home.n*, and *time.n* in your LDA model? Does this model capture lexical similarity better than the tf-idf and word2vec spaces? [5 marks]

(j) Complete the function `get_topic_words` to retrieve the words contained in a given topic. Going through the topics learned by your LDA model, are there any meaningful ones which you can identify? (Hint: be cautious of very frequent words that might negatively affect the quality of topic words retrieved) [5 marks]

Question 2 [50 marks]

Complete the code for functions given in *question2.py* where stated. Again, do not use `numpy` or `gensim`, unless you're explicitly asked to. Give answers for questions in your *answers.pdf* file.

In this question, we will perform the actual lexical substitution task using the functions and models you developed for Question 1 above. The test sentences are provided in `data/test.txt` in json format (Json format can be loaded into a python dictionary using `json.loads()`). For each sentence (json key: `sentence`) in the file, a target word is provided which has to be replaced by its most similar word (the target word's key is `target_word` and its position in the sentence is in the key `target_position`). The substitutes for each target word are provided in the file, `data/test_thesaurus.txt`. The format of this file is:

```
phone.n    sound.n telephone.n telecom.n speech.n
```

where the first word (e.g., `phone.n`) is the target word and the following words (e.g., `sound.n telephone.n telecom.n speech.n`) are the substitutes.

The general architecture of Dinu and Lapata (2010) is as follows: Given a sentence s and a target word t, we define its context words C_s as the words left and right of the target word at a distance ≤ 5. If the context word is not present in the vocabulary (i.e., in the file `data/vocabulary.txt`), ignore it.[5] For each c in C_s, a context-sensitive

[5] This will result in cases with less than 10 context words, which is fine.

vector $\mathbf{v}(\mathbf{t},\mathbf{c})$ is created. Then for a substitute word w its substitution score is computed by aggregating the similarity scores of each context-sensitive vector $\mathbf{v}(\mathbf{t},\mathbf{c})$ with the substitution's word vector $\mathbf{v}(\mathbf{w})$:

$$score(w|t,s) = \sum_{c \in C_s} cosine\_similarity(\mathbf{v}(\mathbf{w}), \mathbf{v}(\mathbf{t},\mathbf{c}))$$

We choose the highest scoring word w', as the best substitution.

$$w' = \arg\max_{w} score(w|t,s)$$

Based on Dinu and Lapata (2010), we will develop three substitution models, and use them with tf-idf, word2vec, and LDA to calculate the best substitutes for our test sentences. The output will be written to a number of files, with each file containing one line for every predicted substitution. The output format should be:

```
targetword.pos sentid :: substitutionword
```

A sample output is shown in the file `data/sample_output.txt`. If your model fails to predict for any reason, leave the prediction empty (see lines 3, 8, 12 etc. in the sample output file). Note that this line needs to end with a space!

(a) Complete the functions `addition` and `multiplication` which take two vectors as arguments and add/multiply them (see equations (5) and (6) of Mitchell and Lapata (2008)). Make sure they work for both sparse and full vectors. [5 marks]

(b) Complete the `addition` part in the function `best_substitute`. This part should use vector addition to compute context-sensitive vectors:

$$\mathbf{v}(\mathbf{t},\mathbf{c}) = addition(\mathbf{v}(\mathbf{t}), \mathbf{v}(\mathbf{c}))$$

The output is the best substitution word for the target word in a test sentence.

Compute the best substitution words for all test sentences in tf-idf and word2vec space, and write the output to `tf-idf_addition.txt` and `word2vec_addition.txt`, respectively. [7 marks]

(c) Complete the `multiplication` part in the function `best_substitute`. This part should use vector multiplication to compute context-sensitive vectors:

$$\mathbf{v}(\mathbf{t},\mathbf{c}) = multiplication(\mathbf{v}(\mathbf{t}), \mathbf{v}(\mathbf{c}))$$

The output is the best substitution word for the target word in a test sentence.

Compute the best substitution words for all test sentences in tf-idf and word2vec space, and write the output to `tf-idf_multiplication.txt` and `word2vec_multiplication.txt`, respectively. [7 marks]

(d) Complete the functions `prob_z_given_w` and `prob_w_given_z`, which calculate the probability of a topic z given some word w, and vice versa. Then complete the LDA part in the function `best_substitute`. Use LDA to compute context-sensitive vector $\mathbf{v}(\mathbf{t},\mathbf{c})$ and any other vector $\mathbf{v}(\mathbf{w})$. In LDA, the context sensitive vectors are calculated according to equation (2) of Dinu and Lapata (2010):

$$\mathbf{v}(\mathbf{t},\mathbf{c}) = (P(z_0|t,c), P(z_1|t,c), \ldots P(z_{99}|t,c))$$

where $P(z_i|t,c)$ is calculated approximately using:

$$P(z_i|t,c) \approx \frac{P(z_i|t)P(c|z_i)}{\sum_k P(z_k|t)P(c|z_k)}$$

For more details, refer to Section 3 (equation (4)) of Dinu and Lapata (2010). You need not calculate the denominator since it is a common factor for all the dimensions of $\mathbf{v}(\mathbf{t},\mathbf{c})$. Using this LDA model, output the best substitution of target words in all sentences. Write the output to `output_lda.txt`.

Hint: Should you receive a UnicodeWarning, you may want to consider using Gensim's `util.any2unicode` function. [15 marks]

(e) In this step, we will evaluate the models. We will use the standard metrics and evaluation script provided by the lexical substitution task organizers. Use the command:

```
perl score.pl your_output_file data/test_gold_answers.txt -t best
```

to compute evaluation scores of your model. Report your precision and recall scores (not the mode precision/recall).[6] [6 marks]

(f) How do tf-idf and word2vec compare against each other when using addition and multiplication to determine the best substitution words? How does LDA perform, and which model is best overall? Discuss your results. [10 marks]

References

Georgiana Dinu and Mirella Lapata. Measuring distributional similarity in context. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1162–1172, Cambridge, Massachusetts, October 2010.

Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *Proceedings of ACL-08: HLT*, pages 236–244, Columbus, Ohio, June 2008.

[6] For more details on the evaluation metric, refer to Section 4.1 of http://nlp.cs.swarthmore.edu/semeval/tasks/task10/task10documentation.pdf