University of Toronto, Department of Computer Science
CSC 2501/485F—Computational Linguistics, Fall 2015

Assignment 5

Due date: 13h10, Wednesday 9 December 2015, on CDF.
This assignment is worth 15% of your final grade.

- Please type your answers in no less than 12pt font; diagrams and structures may be drawn with software or neatly by hand.
- Any clarifications to the problems will be posted on the course website and the CDF forums. You will be responsible for taking into account in your solutions any information that is posted there, or discussed in class, so you should check the page regularly between now and the due date.
- This assignment must be done individually; the work that you submit must be your own. You may, and in fact *should*, discuss your work with others in general terms. But you should avoid looking at one another's work or talking about it in detail. If you need assistance, contact the instructor or TA.

Introduction

This assignment has been modified from its original version. It has been formatted (reduced) to fit this your schedules.

In this assignment we take a superficial look at word2vec[1], specifically the Python implementation in Gensim[2]. There are 3 quick sections:

1. Experiment with training parameters and make some predictions.
2. Design a very simple sentiment analysis engine.
3. Implement a smoothed softmax function.

The Gensim module has been downloaded locally on the CDF machines. To access it, be sure to run the following command either before you run Python, or in your startup script:

```
setenv PYTHONPATH /u/frank/csc485/A5/modules/
```

Naturally, if you already define `PYTHONPATH`, simply add the indicated directory to it.

1. Experiment with training (15 pts.)

In this experiment, we will simply use a small set of text data used for sentiment analysis and learn vector representations of the words therein using different parameterizations of the learning algorithm. Specifically, using `/u/frank/csc485/A5/data_utils.py`, load the data thus:

```
from data_utils import StanfordSentiment
from gensim.models import word2vec
import logging

logging.basicConfig(format='%(asctime)s : \
                            %(levelname)s : %(message)s', \
                            level=logging.INFO)
SS = StanfordSentiment()
sentences = SS.sentences()
```

Once the sentences are loaded, iterate over all 72 permutations of the following variables:

feature	values	description
sg	$[1,0]$	1: skip-gram; 0: cbow
size	$[2,50,200]$	the dimensionality of the feature vectors
window	$[1,2,5]$	the maximum distance between the current and predicted words
min_count	$[1,5]$	the minumum number of times a word must occur for it to be part of the model
iter	$[1,5]$	the number of iterations over the corpus during training

[1] `https://code.google.com/p/word2vec/`

[2] `http://radimrehurek.com/gensim/index.html`.
API at `http://radimrehurek.com/gensim/models/word2vec.html`

in the following command:

```
model = word2vec.Word2Vec(sentences, sg=sg, size=size,\
                          window=window, min_count=min_count,\
                          iter=iter)
```

You can save models, if you want, with `model.save(fname)`, given filename `fname`, and you can similarly load saved models with `model=word2vec.Word2Vec.load(fname)`. These files *can* get large, though, so be warned. If you're worried about the time it takes to train these models, don't forget that the `screen` command exists.

1.1 Qualitative analysis (10 pts.)

Word2Vec, and its ilk, are often evaluated in terms of how intuitively related cherry-picked words are to lists of nearby words are. For each of the 72 models trained, find the (up to) 10 words most similar to the following sentiment-related words:

- laugh, love, enjoy
- yawn, hate, uncomfortable

You can obtain lists of these most-similar words by using the following command, e.g.:

```
existentialAngst = model.most_similar('laugh', topn=10)
```

You can report these in a single Excel spreadsheet or comma-separated file. Each row will represent a model (and hence there will be 72 rows, plus whatever headers you'd greatly enjoy to prepend). The first 5 columns will indicate the particular values chosen for the 5 features listed above. Thereafter, report up to 10 words that are most similar to those 6 above; each of the 6 sets of related words can be enveloped with curly brackets (e.g., '{' or '}').

In a report of between 100 and 200 words, describe any trends you observe in these data. Naturally, you will encounter some very strange matches, which is largely due to the small amount of data, but there are other trends that relate to the parameters you provide. Focus in particualr on the 'skip-gram vs. cbow' differences (or lack of difference) and explain them if you can. Remember you can cite external sources.

1.2 Pseudo-quantitative analysis (5 pts.)

Although Word2Vec is trained in an unsupervised manner, it can still be evaluated semi-objectively. Usually this depends on the application (e.g., whether or not one word model or another results in better speech recognition accuracy, better part-of-speech tagging, and so on).

We spoke in Lecture 10 about the simplicity of drawing analogies within word vectors (e.g., *dog* is to *dogs* as *octopus* is to *octopodes*). We can use some hand-curated date to determine a rough estimate of the 'accuracy' of the system thus:

```
model.accuracy('/u/frank/csc485/A5/data/questions-words.txt')
```

This will provide you with accuracies in roughly the following format, across multiple categories:

```
2015-10-20 10:24:22,453 : INFO : family: 2.8% (2/72)
2015-10-20 10:24:22,934 : INFO : gram1-adjective-to-adverb: 0.4% (1/240)
2015-10-20 10:24:23,057 : INFO : gram2-opposite: 0.0% (0/56)
```

```
2015-10-20 10:24:23,543 : INFO : gram3-comparative: 1.2% (3/240)
2015-10-20 10:24:23,639 : INFO : gram4-superlative: 4.8% (2/42)
2015-10-20 10:24:23,958 : INFO : gram5-present-participle: 0.0% (0/156)
2015-10-20 10:24:23,999 : INFO : gram6-nationality-adjective: 0.0% (0/11)
2015-10-20 10:24:24,322 : INFO : gram7-past-tense: 0.0% (0/156)
2015-10-20 10:24:24,447 : INFO : gram8-plural: 0.0% (0/56)
2015-10-20 10:24:24,673 : INFO : gram9-plural-verbs: 0.0% (0/110)
2015-10-20 10:24:24,673 : INFO : total: 0.7% (8/1139)
```

For the 72 models from part 1.1, produce another spreadsheet that reports these 11 percentages (including total). Report in fewer than 200 words on any trends you observe and make thoughtful suggestions on how to improve these accuracies (beyond the obvious, that we need more data).

Bonus: If you want to be fancy, you can choose a few of those parametric dimensions and graph the average total accuracies. For instance, you could generate a bar graph showing the total accuracy given `sg=1` and `sg=0`, averaged over all other features, in each case. If you want to be *really* fancy, you can actually attempt to implement improvements (including new data, new code, or different parameters). Document and submit anything of which you are proud.

2. Simple sentiment analysis engine (10 pts.)

We're now going to train a (ridiculously) simple sentiment analysis engine using word vectors from part 1. Use (and identify) the model from part 1.2 that gives you the highest overall total accuracy in that section. This distribution of word vectors will be fixed throughout these experiments. We will now use a mapping between words and their 'sentiment', which you get from this file:

`/u/frank/csc485/A5/data/stanfordSentimentTreebank/sentiment_labels.txt`.

For this secton, we want to produce a sentiment classifier that will make guesses as to the sentimental value of words it has never seen before given words that (ought to) be close to them. Here, we'll simply use the WEKA toolkit[3] to do our machine learning. Consult the Appendix as to how you should invoke WEKA.

First, you will need to write your word vectors to an `arff` file (see the Appendix for its format). All you have to do is walk through the rows in:

`/u/frank/csc485/A5/data/stanfordSentimentTreebank/dictionary.txt`,

select *only rows with one word*, get the associated word vector from your selected model (e.g., using `model['computer']` for the word *computer*), and then append the **discretized** sentimental value of that word from the `sentiment_labels.txt` file. How do you discretize the continuous sentimental value from `sentiment_labels.txt`? Like this:

range	[0,0.2]	(0.2,0.4]	(0.4,0.6]	(0.6,0.8]	(0.8, 1.0]
class	0	1	2	3	4

Remember, each row in the arff is a word, so if your word vectors are H-dimensional, you'll have $H+1$ elements on each row, including the word's sentment. WEKA automatically runs

[3] `http://www.cs.waikato.ac.nz/ml/weka/arff.html`

cross-fold validation with the `-x` command (use 10 folds, please). That means that it will try to guess the sentimental value of a word given models trained on other words. Simply report the accuracies you obtain, or provide a screengrab of the output of WEKA.

Bonus: Experiment with different classifiers. Implement cross-fold validation yourself (leave-one-out may be preferred in the Real World, but may be overkill for this assignment). If you want to be super fancy, you can make inferences on whole sentences using the syntactic trees also provided among the data on CDF. Report your results in a scientitic manner.

3. Smoothed soft-max (5 pts.)

We spoke (very) briefly in class about smoothing conditional probability functions. In bigrams, we represent the probability of a word w_t following a word w_{t-1} with this representation:

$$P(w_t \,|\, w_{t-1}) = \frac{P(w_{t-1}, w_t)}{P(w_{t-1})} \approx \frac{Count(w_{t-1}, w_t)}{Count(w_{t-1})},$$

where $Count(\cdot)$ is the number of times the argument appears in the training corpus. However, that's not cool because, like, what if you're reading some test data or something and, like, you see a bigram whose first word you've never seen before and you're all like *"oh man, I'm going to determine the probability of this bigram"* but then you're like, *"oh no, that will give me* $0/0$ *probability and I can't handle that in my life right now."*. Well, the normal trick is to pretend you've seen things more often than you have, so we have:

$$P_{smoothed}(w_t \,|\, w_{t-1}) \approx \frac{Count(w_{t-1}, w_t) + \delta}{Count(w_{t-1}) + \delta ||V||},$$

where δ is an empirical parameter, usually between 0.0001 and 0.1, and $||V||$ is the size of your vocabulary V. When $\delta = 1$, this is called 'Laplace smoothing', by the way.

In Lecture 10 we defined the 'softmax' function where a context (outside) word w_o is related to the current (inside) word w_i thus:

$$P(w_o \,|\, w_i) = \frac{\exp(V_{w_o}^\top v_{w_i})}{\sum_{w=1}^{||V||} \exp(V_w^\top v_{w_i})},$$

where v_w and V_w are the 'input' and 'output' vectors of the model (i.e., each word is represented by two vectors – once as the current word and once as the context word).

Implement a Python function that merely takes a matrix θ (after slide 26 in Lecture 10) of $2 \times ||V||$ rows, each of dimension H and implements the softmax function. But there's more! Your function should also take a parameter δ so that you can smooth these probabilities. You are responsible to determine how to incorporate δ into the softmax function, as long as the result is a valid conditional probability function in which every pair $\{w_o, w_i\}$ is $0 \leq P(w_o \,|\, w_i) \leq 1$ and $\sum_{w \in V} P(w \,|\, w_i) = 1$, given any w_i.

Bonus You can evaluate your functions using actual vectors from Word2Vec. As this is for your own amusement, you can also take the time to figure out how.

What to submit electronically Please submit your code, your spreadsheets, and your reports for all three parts. Submit the files using the `submit` command on CDF:

```
% submit -c <course> -a A5 <filename-1>...<filename-n>
```

where `<course>` is `csc485h` for undergraduates and `csc2501h` for graduate students, and `<filename-1>` to `<filename-n>` are the *n* files you are submitting. Make sure every file you turn in contains a comment at the top that gives your name, your login ID on CDF, and your student ID number.

Appendix: Using WEKA

WEKA is available on CDF in */u/frank/csc485/A5/WEKA*.

1. Classifier command line

The general syntax for running WEKA from the command line is:

```
java -cp *weka.jar path* *weka classifier* *classifier options*
```

The path to the jar file on CDF is: `/u/frank/csc485/A5/WEKA/weka.jar`.

You only need to use the `weka.classifiers.functions.SMO` classifier (for SVMs). The key options are:

-t *arff file* (Choose the training file)
-*x* *X* (Use cross-validation, X is the number of folds)
-o (Do not output the classifier. Use this option for your outputs!)

2. Information gain attribute selector

The options for running WEKA's information gain attribute selector are different than the classifiers and a bit more complex; ideally, attribute selection is done through the WEKA GUI (and you may do so, see below). Since there is no variation in the parameters, however, we can simply provide you with a sh script that will do the job. To run it on one of your arff files and send the output to a file, use this command:

```
sh /u/frank/csc485/A5/WEKA/infogain.sh *arff file* > *output*
```

3. WEKA GUI

WEKA includes a GUI for quickly testing various options associated with classification. You may use the GUI for early testing and additional exploration as part of the bonus. If you are on a CDF lab computer or logged in using NX, the GUI is available using the following command:

```
java -jar /u/frank/csc485/A5/WEKA/weka.jar
```

Select the explorer, and open your arff file using the open file button. Then go to the Classify tab to carry out a classification experiment, or Select Attributes to do feature selection. The options which can be selected otherwise correspond to those available at the command line. See
`http://www.cs.waikato.ac.nz/ml/weka/index_documentation.html` for more information.

Table 2: Example `weather.arff` for WEKA

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```