CS 477/577 Natural Language Processing

Homework 0: Introduction to the NLP Programming Enviroment

Instructor: Dragomir Radev *Due: January 23rd, 2019*

I. Introduction

This assignment has two goals:

- give you some background needed for the class and further assignments
- help you get familiar with the programming environment

II. Environment Setup

While we encourage you to work locally if you choose, we have put instructions to work on the Zoo cluster here. For more information about the cluster and how to SSH, visit `http://zoo.cs.yale.edu/newzoo/`. Assuming that you have SSH'ed to your home directory on Zoo, run the following commands.

Step 1: Set up a hidden folder.

You will need to set up a hidden folder on Zoo so that only you and the course staff can see your work, but your classmates cannot. To do this, SSH into Zoo and follow the instructions below. Start in your home directory:

```
cd
```

You will have received (by Monday) an email from Drago that contains the pin you have been assigned for the course. Please make sure to consistently use this pin throughout the semester so that the course staff can access your assignments for grading.

Run the script with your pin:

```
/home/classes/cs477/bash_files/make_hidden.sh YOUR_PIN
```

That's it! Your home directory should now have a folder called hidden. To check that everything worked, run the following command:

ls -l [Note that this is a lowercase L, not the digit 1]

You should see a line that looks something like this:

```
drwx-wx--x  3 netid netid  4096 Jan 16 23:13 hidden
```

Note the dashes in drwx-wx–x. If you have an r in place of any of these dashes, something went wrong, and you should try rerunning the script. If you look inside the hidden folder, you can see a folder

with your pin number, but no other student will be able to read what's in your hidden folder. This is important so that other students cannot copy your code.

Step 2: Create a symbolic link to the NLTK data for this class.

Go to your home directory:

```
cd
```

Create a symbolic link to NLTK data:

```
$ ln -s /home/classes/cs477/data/nltk_data/ ~/nltk_data
```

Step 3: Copy the homework files to your hidden directory under Homework0 folder.

```
$ cp -r /home/classes/cs477/assignments/Homework0 ~/hidden/<YOUR_PIN>/
```

Navigate to the homework folder, and list its contents:

```
$ cd ~/hidden/<YOUR_PIN>/Homework0
$ ls
```

You should see the following files:

- A.py
- B.py
- correct_outputA.txt
- correct_outputB.txt
- input.txt
- squirrel_girl.txt
- HW0_Introduction.ipynb

Step 4: Launch a virtual environment for this assignment

```
$ source /home/classes/cs477/venvs/hw0/bin/activate
```

Step 5: Open Jupyter Notebook on your machine

Jupyter notebooks are a tool to organize code as well as markdown and figures into an easily-readable format. You can read more about jupyter notebooks at this link. The Jupyter notebook will help guide you through the assignment as well as provide you with additional information and resources, so we highly recommend following the below steps to launch the notebook we have provided. **NOTE: you will not be submitting the Jupyter notebook, so make sure to follow the instructions for submission.**

After you have launched the virtual environment, run the following command (only needs to be run once):

```
$ python -m ipykernel install --user --name=py36env
```

Run the following command with your own port number (the example 8889 may already be in use) on Zoo:

```
$ jupyter notebook --no-browser --port=[your/port/number/e.g./8889]
```

After you run this command, you will something like:

```
...to login with a token: http://localhost:[your/port]/?token=4e***ad&token=4e***ad
```

You will need to copy whatever appears after the last `token=` into your local browser later.

Open a new terminal window on your local machine, SSH into the remote machine again using the following options to set up port forwarding:

```
$ ssh -N -L localhost:8888:localhost:[your/port] [your/netid]@node.zoo.cs.yale.edu
```

Access the remote jupyter server via your local browser. Open your browser and go to:

```
$ localhost:8888
```

You need to copy the token mentioned above in order to access the notebook. Please refer to this link for more details.

Once you have opened the notebook, make sure to select the kernel py36env. This is done within the notebook by selecting Kernel in the top menu, Change kernel in the resulting dropdown menu and selecting py36env.

III. Part A: Warm up with Linux and Python Basics

In this part of the assignment, you will learn a few handy Linux commands and play with a Python script. But wait! You've already learned some handy Linux commands, just by setting up your environment. Let's review those, first:

cd	Change directory. For example, run cd ~/hidden.
cp	Copy. If you want to copy a whole folder, you need to use the r option.
grep	Search for text. You need to tell it what to search for and where; for instance, grep import A.py will show you all the times "import" appears in A.py. Use the r option to search a directory instead of just one file. A shortcut to the current directory is a single dot, so grep – r resources . searched all of the files in the current directory.
ls	List a directory's contents. You can run it without any arguments to list the contents of the current directory, or give it a path to list the contents of a different directory. For example, if you're in your home directory, running ls ~/hidden/Homework0 will list the files in your Homework0 folder.
mkdir	Make a new directory

This is only the tip of the Linux iceberg. If you'd like to learn more Linux commands, you can find plenty of Linux cheat sheets like this `http://files.fosswire.com/2007/08/fwunixref.pdf` online. If you're looking for more detail, nixCraft `http://www.cyberciti.biz/` is often helpful.

Time-saving tip:

Lots of commands use the names of folders or files as arguments. Rather than typing these out, you can use tab completion to make Linux finish your thought for you. Try this: cd to your home directory, then start typing cd h, and hit the tab key. It should automatically expand to cd hidden/, and if you hit tab again, it will fill in your pin!

Now, let's move on to our files. If you open the code of A.py, you can see what's happening: Lines 1 and 2 import some modules —- pre-written code. Line 4 defines a variable (in this case, it's a string) called greeting, and line 5 prints it to the screen. In line 7, we use the nltk module's word_tokenize method to make a list of all of the tokens–words or punctuation in our greeting string. (Python lists are kind of like arrays that automatically grow or shrink as needed. Like an array, you can iterate through it or index into it.) We then use the for loop in lines 9 and 10 to print each token on its own line. Try running on your hw0 directory:

```
python3 A.py
```

You should get the output:

```
Hello, world!
The tokens in the greeting are
Hello
,
world
!
```

That's some really boring output, isn't it? Let's make it more interesting by replacing "Hello, world!" with text the user will provide on the command line. **First, modify Line 4 of A.py so that the command**

```
echo "This is a different greeting" | python3 A.py
```

yields

```
This is a different greeting


The tokens in the greeting are
This
is
a
different
greeting
```

Hint: If you're having trouble, take a look at this StackExchange Q&A: `http://stackoverflow.com/questions/11109859/pipe-output-from-shell-command-to-a-python-script`

What just happened? You used a pipe (|) to use the output of the first command as input to the second. Your first command was an echo, which just outputs the same thing you type in.

What if you already have a file with the text you want to use as input to your script? For example, suppose your GSI had already typed up the Squirrel Girl[1] theme song for you, and you didn't want to retype the whole thing. One thing you can do is use cat <filename>, which outputs the text of the file:

[1] `https://en.wikipedia.org/wiki/The_Unbeatable_Squirrel_Girl`

```
cat squirrel_girl.txt
```

should print the Squirrel Girl lyrics to the screen. You can also run A.py on these lyrics by piping the output of your cat command into your python command, like so:

```
cat squirrel_girl.txt | python3 A.py
```

There are a lot of repeated words in this song, aren't there? I wonder if there are more repetitions of the word squirrel or girl. Let's find out! Add the following line to the end of A.py (it should be just one line):

```
print(f"There were {squirrel} instances of the word 'squirrel'
and {girl} instances of the word 'girl.'")
```

Now, modify the for loop to keep a count of the number of times "squirrel" appears and the number of times "girl" appears. Make the count case insensitive.

- Hint #1: Make sure you're initializing both variables (squirrel and girl) to 0 before the loop!
- Hint #2: The old standby "++" that you may know from other languages won't work in Python, but you can still use "+= 1"
- Hint #3: `https://docs.python.org/2/library/stdtypes.html#str.lower`

Half your grade for this assignment will be based on whether A.py is producing the right output, so you'd probably like to confirm that the output is correct. We have given you the correct output in correct_outputA.txt. You can check that your output matches ours by saving yours to a file, then using the diff command:

```
cat squirrel_girl.txt | python3 A.py > outputA.txt
diff outputA.txt correct_outputA.txt
```

where the > operator in the first line writes the text that would normally be printed to stdout into the file outputA.txt instead. If everything is working, the second line should return with no output; this means there were no differences between the two files. If you accidentally replaced the first line of the song with "This line shouldn't be here." and then ran diff, you would see

```
1c1
< This line shouldn't be here.
---
> Squirrel Girl, Squirrel Girl!
```

instead. This would show you exactly where your output was wrong, which should help you pinpoint the problem in your code.

IV. Part B: Word Similarity

Word similarity can play an important role in information retrieval. For instance, suppose a user asked for information about birds, and your system had to decide which of the following passages to return:

Stoats (Mustela erminea) were introduced into New Zealand to control rabbits and hares, but are now a major threat to the native bird population. The natural range of the stoat is limited to parts of the Northern Hemisphere. Immediately prior to human settlement, New Zealand did not have any land-based mammals apart from bats, but Polynesian and European settlers introduced a wide variety of animals[2]

Or

Eagles tend to be large birds with long, broad wings and massive feet. Booted eagles have legs and feet feathered to the toes and build very large stick nests. Ospreys, a single species found worldwide that specializes in catching fish and builds large stick nests. Kites have long wings and relatively weak legs. They spend much of their time soaring. They will take live vertebrate prey, but mostly feed on insects or even carrion.[3]

Clearly, the second passage is more relevant, even though “bird" appears once in the first passage and “birds" appears once in the second. As humans, we know that words like “eagles" and “ospreys" are similar to “bird," while words like “stoats" and “rabbits" are not very similar to “bird."

In this part of the assignment, you will compare three word similarity measures to determine which agrees the most with human ratings of similarity. The three methods are Lin similarity, Resnik similarity, and a vector-based similarity measure. Lin similarity and Resnik similarity are both based on WordNet (`https://wordnet.princeton.edu/`), a hand-built taxonomy of English words. The vector-based method represents words with GloVe vectors that were automatically learned by a system that looked at a corpus of billions of words and measures how similar the words are by how close together their vectors are.[4] In the vector-based method, the cosine between vectors often represents closeness. You will also see that word vectorization is important for neural network based algorithms as well in later lectures. Such word vectorization is also called word embedding in the literature since it embeds a discrete space with one dimension per word into a continuous space with lower dimension.

B1: Parse the input file.

First, implement parseFile. This function will be given the name of a file that contains a word pair and a similarity score on each line. To see an example of the format of the file, look at input.txt. Your function should read in the file and return an ordered dictionary of (word1, word2): score entries. Make sure that you represent your scores as floats.

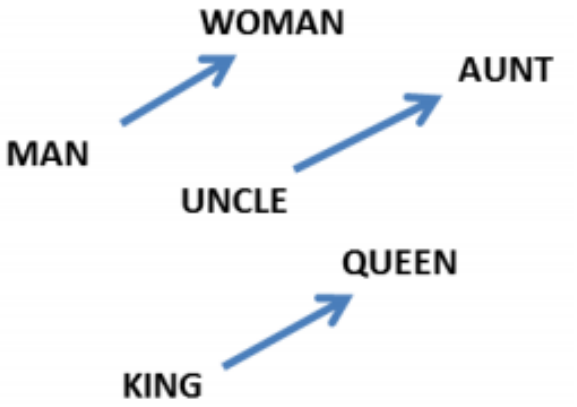
Hint: You can find out more about file I/O in Python here: `https://docs.python.org/3.6/tutorial/inputoutput.html#reading-and-writing-files`

[2] From `https://en.wikipedia.org/wiki/Stoats_in_New_Zealand`

[3] From `https://en.wikipedia.org/wiki/Bird_of_prey`

[4] You can learn more about these methods in the following papers:

- Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. Proceedings of the 14th international joint conference on Artificial intelligence (IJCAI’95).
- Dekang Lin. An Information-Theoretic Definition of Similarity. In Proceedings of the Fifteenth International Conference on Machine Learning (ICML ’98)
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.



B2: Use WordNet to measure similarity

Next, implement linSimilarities and resSimilarities using NLTK. These functions are given a list of word pairs and a WordNet IC corpus, and each should return a dictionary of (word1, word2): score entries. For this exercise, represent each word in the pair as the first synset in the list of noun synsets related to the word. If no noun synset is found for a word in the pair, represent each word as the first synset in the list of verb synsets related to the word. Note that the NLTK Lin scores are between 0 and 1; these will need to be scaled by a factor of 10 to be comparable to the human similarity scores.

Hint: Use the wn.synsets() method.
Hint: You can find out more about NLKT's similarity functions here: `http://www.nltk.org/howto/wordnet.html`

B3: Use the vector-based similarity measure

The vector-based similarity measure represents words with vectors (word embeddings) so that similar words are close together in the space. Word embeddings become the most popular way to represent words especially when deep learning models are introduced to solve many NLP tasks (these models usually use word embeddings of words as inputs). We're going to implement a neural network using word embeddings in this class later.

Figure 1 shows the a word embedding space introduced by Mikolov et al. (2013a).

You can see not only these words of humans are close together, but also their relations are captured. For example, if you take the vector of "woman" minus the vector of "man," the resulting vector is the same as 'Vector("aunt") −Vector("uncle")'.

You can use PCA to visualize the embedding space by yourself. PCA is a dimension reduction technique useful for visualizing high dimensional data in low dimensions (usually 2). The following code creates a scatterplot of some word2vec embeddings in 2D space. Please check out the HW0_Introduction.ipynb to create the Figure 2.

Look at the main function. Notice the following line that is commented out:

```
model = gensim.models.KeyedVectors.load_word2vec_format(RESOURCES+'glove_model.txt', binary=False)
```

This line will load a pre-trained vector-based model into memory[5]. Uncomment them. Before you do

[5] Wondering why the method is called Word2Vec when we're actually using GloVe vectors? Word2Vec and GloVe produce extremely similar vectors, but GloVe is a little newer. Gensim doesn't officially support GloVe yet, but by inserting a header into the GloVe file, we trick Gensim into being able to use all the Word2Vec functions with our GloVe vectors

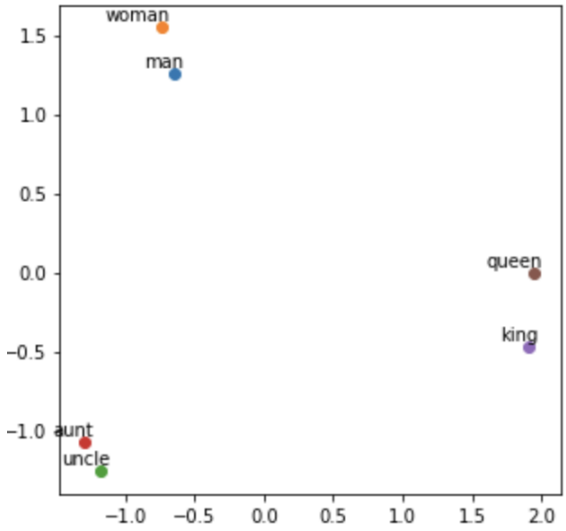


Figure 1: PCA visualization of word embedding space

anything else, try running your code again, and notice how the runtime changes.

Now, implement vecSimilarities. The usual list of word pairs and the pre-trained model will be passed in to your function. Like the other functions, it should return a dictionary of (word1, word2): score entries. You can use Gensim to get the cosine similarity between the vector for word1 and the vector for word2; you'll need to multiply by ten like you did with the WordNet functions. Note you may need to convert the word to lower case.

Hint: See what the model can do here: `https://radimrehurek.com/gensim/models/word2vec.html`

B4: Save your output and check your work

Use the same technique you saw in Part A to save the output of Part B to outputB.txt. We have provided correct_output_B.txt, so you can diff your output against ours.

V. Part C: Comparing Numpy with Pytorch

We will use Pytorch in most of our assignments in this class. We suppose that you are already familar with Numpy. In this section, we implement two same simple networks in both Numpy and Pytorch so that we can get some senses of differences between them. Check out more details in the Pytorch official tutorial

At its core, PyTorch provides two main features: an n-dimensional Tensor, similar to Numpy but can run on GPUs, and automatic differentiation for building and training neural networks

The Pytorch official tutorial uses a fully-connected ReLU network as a running example. The network will have a single hidden layer, and will be trained with gradient descent to fit random data by minimizing the Euclidean distance between the network output and the true output.

Numpy provides an n-dimensional array object, and many functions for manipulating these arrays. Numpy is a generic framework for scientific computing; it does not know anything about computation graphs, or deep learning, or gradients. However we can easily use Numpy to fit a two-layer network to random data by manually implementing the forward and backward passes through the network using Numpy operations.

Numpy is a great framework, but it cannot utilize GPUs to accelerate its numerical computations. For modern deep neural networks, GPUs often provide speedups of 50x or greater, so unfortunately Numpy won't be enough for modern deep learning. To run a PyTorch Tensor on GPU, you simply need to cast it to a new datatype. However, we aren't using GPUs on the Zoo machines.

In the Jupyter notebook we introduce the most fundamental PyTorch concept: the Tensor. A PyTorch Tensor is conceptually identical to a Numpy array: a Tensor is an n-dimensional array, and PyTorch provides many functions for operating on these Tensors. Behind the scenes, Tensors can keep track of a computational graph and gradients, but they are also useful as a generic tool for scientific computing.

Please go over this section in HW0_Introduction.ipynb to get a basic idea of the differences between Numpy and Pytorch.

VI. Part D: Advanced PyTorch

For later assignments we will use PyTorch as a higher level library which provides functions for automatic differentiation, so we do not need to perform backpropagation by ourselves. Additionally, for complicated neural networks, we do not need to write the entire architecture from scratch, as PyTorch provides many fundamental models which we can combine and modify. (This section is based on this tutorial)

An example of such a model is the Long Short-Term Memory Network (LSTM), which you can read more about here. LSTMs will be covered later in the class, but for now you just need to know that they are used for sequence modeling by keeping track of a hidden state which is dependent upon previous inputs in the sequence. Sequence modeling is essential for NLP, as this defines some kind of temporal dependence in the input sequence.

In most cases, words and labels must be converted to indices in order to be fed into the network. These indices are then converted to tensors by using the prepare_sequence function. In Part D of HW0_Introduction.ipynb, we will look at part of speech training; for each word we want to find its labeled part of speech. We will use some toy data to train a simple model.

Please also go over this section on HW0_Introduction.ipynb to see the example of how to define and train a neural model using PyTorch's high-level libraries.

Submit your assignment.

For this and all other assignments, you will submit by making sure all required files are in the appropriate homework folder in your hidden folder on Zoo. It is fine to have other files in the directory as well. We use the timestamp of the required files on Zoo to determine if you submitted on time, so please do not modify these files after the assignment deadline. Please note that we will only grade files on Zoo.

For this assignment, please turn in the following files in ~/hidden/<YOUR_PIN>/Homework0:

- A.py
- B.py
- outputA.txt
- outputB.txt

Grading Criteria

This assignment is worth two points.

Part A:

One point if running cat squirrel_girl.txt | python3 A.py produces output identical to correct_outputA.txt.

Part B:

One point if running python3 B.py produces output identical to correct_outputB.txt.

Late Day Policy

10% off for each day. After three days, the assignment will be given a score of zero.