Assignment 1

CMSC 473/673 — Introduction to Natural Language Processing

Due Wednesday September 4th, 2019, 11:59 PM

<table>
<thead>
<tr>
<th>Item</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assigned</td>
<td>Wednesday August 28th, 2019</td>
</tr>
<tr>
<td>Due</td>
<td>Wednesday September 4th, 2019</td>
</tr>
<tr>
<td>Topic</td>
<td>Warmup with Counting and Basic Probabilities</td>
</tr>
<tr>
<td>Points</td>
<td>55</td>
</tr>
</tbody>
</table>

In this assignment you will step through some introductory NLP techniques.

You are to complete this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>25</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Don’t let this handout’s length be deceiving: the handout may be lengthy, but think of it as both a tutorial and assignment. There is a lot of explanation and hints to help you along.

What To Turn In  Turn in a writeup in PDF format that answer the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit  Submit the assignment on the submission site:

[https://www.csee.umbc.edu/courses/undergraduate/473/f19/submit](https://www.csee.umbc.edu/courses/undergraduate/473/f19/submit)

Be sure to select “Assignment 1.”
Questions

1. (25 points) Hal Daumé III has a very nice “refresher” tutorial called “Math for Machine Learning:”
   http://www.umiacs.umd.edu/~hal/courses/2013S_ML/math4ml.pdf Where indicated, some of the following questions are taken from that primer. I encourage everyone to read the
   primer, but especially if you have difficulty answering the following questions.
   
   (A) [based on Exercise 1.2] Compute the derivative of the function \( f(x) = \exp(-\frac{1}{2}(x - \mu)^2) \) with
   respect to \( x \) (assume \( \mu \) is a constant).
   
   (B) [Exercise 1.5] Compute the derivative of the function \( f(x) = \log(x^2 + x - 1) \).
   
   (C) Compute the derivative with respect to \( x \) of the function \( f(x) = \log(\sum_{k=1}^{K} \exp(kx^k)) \), for
   finite, positive, integral \( K \).
   
   (D) Compute the derivative with respect to \( x \) of the function \( f(x) = \log(\prod_{k=1}^{K} \exp(kx^k)) \), for finite,
   positive, integral \( K \).
   
   (E) [Exercise 2] Given \( N \) points \( \{(x_n, y_n)\}_{n=1}^{N} \), compute the (partial) derivative \( \frac{\partial J}{\partial b} \), where
   \( J(m, b) = \sum_{n=1}^{N} ((mx_n + b) - y_n)^2 \). Note that the \( N \) points \( (x_n, y_n) \) can be considered constants. Hint: the
   partial derivative \( \frac{\partial J}{\partial b} \) is a derivative with respect to \( b \), where you treat all other variables as con-
   stant.
   
   (F) [Exercise 3] For \( J(m, b) \) defined on the \( N \) points, as above, compute the values of \( m \) and \( b \) that result in the gradient of \( J \) being the zero vector, i.e., \( \nabla J = (\frac{\partial J}{\partial b}, \frac{\partial J}{\partial m}) \). In this case, finding
   these values minimizes \( J \). You should be able to find closed-form expressions for both \( m \) and \( b \).
   
   (G) Given two \( K \)-dimensional vectors \( u \) and \( v \), compute the dot product \( u^\top v \).
   
   (H) Given the matrix \( A = \begin{pmatrix} 4 & 2 & 0 \\ -1 & 0 & -1 \end{pmatrix} \), compute the values \( A^\top A \) and \( AAA^\top \).
   
   (I) [Exercise 24] For the multivariate function \( f(u, v) = \exp(u^\top v) \), where \( u, v \in \mathbb{R}^K \), compute the
   gradients \( \nabla_u f \) and \( \nabla_v f \).

2. (10 points) Let \( X_i \) be the random variable representing the \( i \)th role of a six-sided die and let \( x_i \) be the
   rolled (observed) value of the \( i \)th roll. Assume that each roll is independent from one another and the
die doesn’t change between rolls. Let \( x_1, \ldots, x_4 \) be the results of rolling that die 4 different times.
Each \( x_i \) can have a value of 1, 2, 3, 4, 5, or 6.
   
   (A) If the die is fair (each of the six sides is equally likely), compute \( p(x_1, x_2, x_3, x_4) \) (the probability of
   observing those four rolls in that particular order).
   
   (B) The expected value of a random variable \( X_i \) is the average the outcomes of \( X_i \), weighted by
how likely each outcome is. Mathematically, we can represent the expected value of \( X_i \) as
   \( \mathbb{E}[X_i] = \sum_k k \cdot p(X_i = k) \), where the sum iterates over all possible values of \( X_i \).
   
   i. Compute \( \mathbb{E}[X_i] \) assuming that each outcome is equally likely.
   
   ii. Compute \( \mathbb{E}[X_i] \) assuming any odd outcome is twice as likely as an even outcome.

3. (20 points) In this question, you’ll be doing some basic counting of words—the “Hello, World” of
NLP.\footnote{We would say that each \( X_i \) is an i.i.d. sample, where i.i.d.
means “independently and identically distributed.”}
In the GL directory
/afs/umbc.edu/users/f/e/ferraro/pub/473-f19/data/ud-treebanks-v2.2
you will find the entire Universal Dependencies dataset. For this question, look at the UD_English-EWT
to folder. In that folder, you'll see a license file, a README file, and three pairs of files. The files we'll
want to be concerned with are the .conllu files. One corresponds to the "training" data, another corre-
responds to the "development" data, and the last corresponds to the "test" data. For this assignment,
do not run your code on the test data.

The .conllu files are tab-separated files, with two exceptions: lines can be blank, and lines can
start with the character #. Lines that are not blank correspond to a particular word in a sentence; lines
that are blank signify the end of the previous sentence, and lines that begin with a # mean that that
line is "commented" and must be ignored (it is not part of the data). If you write code that tries to
interpret lines that begin with a #, you will not receive full credit! The original text of each word
is in the second column of each row. For example, the first sentence has 29 tokenized words. We can
see this by space-separating all of the words for the first sentence of en_ewt-ud-train.conllu:

    Al - Zaman : American forces killed Shaikh Abdullah al - Ani , the
    preacher at the mosque in the town of Qaim , near the Syrian border .

The other columns will be useful later in the semester: they contain information like the lemma for
each word (third column), part-of-speech tags (columns 4 and 5), some types of linguistic features of
the word (column 6), and syntactic information (columns 7-end).

(a) How many sentences are there in the training and development splits? On average, how many
words are there per sentence?

You can use any method you want to compute the number and averages (e.g., Linux commands,
or a small Python or Java program). Turn in what you wrote, or provide the command in your
writeup with a brief explanation of the steps involved.

While it is a good idea to question your data, especially if it looks strange/not what you expected, for
this question you can take these tokens as they are: assume that, for some application, they are useful.
The text you see is called tokenized text. In particular, it is text that has been tokenized, or split into
individual "words," according to a particular specification. You may be surprised that we consider
punctuation as different tokens.

But let's dive into this some more. The individual instances you observe are tokens, where each token
is drawn from a set of types. Using a programming analogy, we can say that word types are like
classes while word tokens are like instances of that class. For example, in the following sentence
there are six types and eight tokens:

    the gray cat chased the tabby cat .

Notice that this computation includes punctuation.

(b) In the training file, how many different word types and tokens are there? Do not perform any
processing that modifies the words. Turn in the code for this, or if done on the command line,
describe how.

---

\[See http://universaldependencies.org/\]
(c) What are some of the most common words, like the twenty, thirty, and fifty most common ones. Discuss what you notice about those common words. Each word should be alphanumerically (lexically) distinct.

(d) In the above question, should all of these items actually be considered distinct? What are some ways that we could group together words? Hypothesize some effects your collapsing would have. You can argue for or against your collapsing method.

*Hint:* There is not a right or wrong answer here. Some collapsing methods may be more appropriate than others, but the question is to think about these methods and what effect they may have. Now, there are simpler answers. In particular, some collapsing methods can be accomplished with simple calls to standard string processing functions. Others could be accomplished with some more advanced processing (e.g., see column 3).

(e) Using the second column, examine some of the least common words. Do words that appear only once look like “standard” words? What about words that appear ten times? Fifty times? What about 100 times?

Regardless of whether these words were standard, are they “reasonable?” Are they items that you would want to be able to talk about as distinct items? From a computational point of view, do you want to spend the computational resources to deal with them?

Argue for or against these items’ reasonableness. If you find them unreasonable, propose a solution. You do not need to implement it.

(f) Now it’s time to look at the development split: this is in the file en_ewt-ud-dev.conllu. How many word tokens are there? How many word types were not seen in the training data? We call these out of vocabulary (OOV) words.

Again, do not perform any processing that modifies the words. Turn in the code for this, or if done on the command line, describe how.