Assignment 1

CMSC 473/673 — Introduction to Natural Language Processing

Due Monday September 10th, 2018, 11:59 AM

<table>
<thead>
<tr>
<th>Item</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assigned</td>
<td>Wednesday August 29th, 2018</td>
</tr>
<tr>
<td>Due</td>
<td>Monday September 10th, 2018</td>
</tr>
<tr>
<td>Topic</td>
<td>Warmup with Counting and Basic Probabilities</td>
</tr>
<tr>
<td>Points</td>
<td>100</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>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>15</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/f18/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) [Exercise 1.2] Compute the derivative of the function \( f(x) = \exp(-\frac{1}{2}x^2) \).
   (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 constant.
   (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) [Exercise 7] Compute the Euclidean, Manhattan, Maximum, and Zero norms on the three vectors \( (1, 2, 3), (1, -1, 0), (0, 0, 0) \).
   (H) Given the matrix \( A = \begin{pmatrix} 4 & 2 & 0 \\ -1 & 0 & -1 \end{pmatrix} \), compute the values \( A^\top A \) and \( AA^\top \).
   (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. (25 points) Let \( x_1 \) be the roll of a six-side die, and let \( x_1, \ldots, x_4 \) be the independent results of rolling that die 4 different times. Each \( x_i \) can have a value of 1, 2, 3, 4, 5, or 6. If \( x_1 = 3, x_2 = 1, x_3 = 6, \) and \( x_4 = 5 \):
   (A) If the die is fair (each of the six sides is equally likely), what is the probability of observing those four rolls?
   (B) What is the average value of the rolls \( x_1 \) through \( x_4 \)?
   (C) Let \( y_i = \exp(x_i) \). What is the average value of \( y_1 \) through \( y_4 \)?
   (D) Let’s say that before we rolled the \( X \) die, we first flipped a fair coin (2 outcomes, where probability of heads equals probability of tails). Call the outcome of each coin flip \( z_i \). If \( z_i = H \) (heads), then we roll a fairly weighted, six-sided die. If \( z_i = T \) (tails), then we roll a six-sided die where the probabilities of rolling each outcome is given by the probability vector \( \pi = \left( \frac{1}{9}, \frac{2}{9}, \frac{1}{9}, \frac{2}{9}, \frac{1}{9}, \frac{2}{9} \right) \) (e.g., rolling a one, three, or five are all equal probability, and rolling a two, four, or six also have all equal probability).

   i. Write the conditional distributions \( p(x \mid z) \). (Hint: You should write out two different distributions.)
ii. The marginal distribution of \( x_i \) is defined as \( p(x_i = k) = \sum_j p(x_i = k, z_i = j) \). What are the marginal probability values \( p(x_1 = 3) \), \( p(x_2 = 1) \), \( p(x_3 = 6) \), and \( p(x_4 = 5) \)?

3. (20 points) In this question, you’ll be doing some basic counting of words—the “Hello, World” of NLP.

In the GL directory
/afs/umbc.edu/users/f/e/ferraro/pub/473-f18/data/ud-treebanks-v2.2
you will find the entire Universal Dependencies dataset. For this question, look at the UD English-EWT 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 corresponds 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). 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 produces

```
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 assignment and 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.

---

1See [http://universaldependencies.org/](http://universaldependencies.org/)
(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.

(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.

4. **(15 points)** Let $V_n$ be the set of word (types) that appear $n$ times in the (training) corpus. For example, $V_1$ are those words that appear just once, $V_2$ are those words that appear twice, and $V_{150320}$ are those words that appear 150320 times. First, pick two different languages from the UD corpus that have both a training set (*train.conllu) and a development set (*dev.conllu). Call these languages A and B.

(a) From the training sets, create a scatterplot of the number of words appearing $n$ times (i.e., $|V_n|$) vs. $n$; this plot must be readable and visually informative. You should compute $V_n^A$ for language A and a separate $V_n^B$ for language B. Discuss what you observe, what (if any) transformations you had to use in order to better analyze the data, and the implications of your observations.

*Hint:* This should not require a lot of memory: no more than 200MB. If it does, check your computation of $V_n$.

(b) Let $U_n$ be the set of out of vocabulary words appearing $n$ times (in the development corpus). Plot $|U_n|$ vs. $n$ and discuss what you observe.

For both parts of this question, turn in both the plot (either as a PDF, PNG or JPG) and any code needed to reproduce the data and plot.
5. **(15 points)** It’s common to say that language follows a power law distribution. Alternatively, you may hear that ‘language is Zipfian.’ In this question, you’ll explore what that means, and examine how Zipfian languages A and B (from the previous question) can be.

First, imagine counting up words and performing a descending sort of the words according to how many times each appeared. That is, words that appear more often appear before words that appear less often—in such a setting, the more common words are said to have a lower rank. The basic Zipf estimate states that the frequency \( f(y) \) of a word \( y \) is inversely proportional to how common it is (its rank \( r \)). Taking logarithms turns this into a linear relationship (in log-log space):

\[
\log f(y) = C - m \log r(y),
\]

where \( C \) and \( m \) are constants.

Explore how well Zipf’s law holds up on some Universal Dependency data. Do this by completing the following for each language A and B:

- First, compute \( f_X(y) \) and \( r_X(y) \) for each word type \( y \) and language \( X \).
- Second, plot \( \log f_X(y) \) vs. \( \log r_X(y) \).
- Third, perform a linear regression (ordinary least squares), which fits a line to provided independent and dependent variables.
- Fourth, include these plots in your writeup and provide a discussion of what you found. What does the slope of the linear regression tell you? What about the coefficient of determination (the \( R^2 \) value)? In your analysis, include a description of your methodology, the results, and an analysis of the results.

Remember to turn in all code needed to reproduce your analysis. Though we’ll have your code, your methodology description should be thorough enough for us to reimplement what you did.

---

\(^2\) To perform this linear regression, you may use any external library. In R, linear regression is available through the `lm` function, from the basic `stats` package (typically preloaded). Linear regression is available in Python through a number of methods, including from `scipy` under `scipy.stats.linregress`. 