Coursework 1 – Foundations of Natural Language Processing

Deadline: 12 noon, Wednesday 14th February, 2024

Introduction

This coursework consists of two parts in which you demonstrate your understanding of fundamental concepts and methods of Natural Language Processing. The coursework contains open questions and programming tasks with Python, mostly using NLTK. This coursework is worth 12.5% of your final mark for the course. It is marked out of 100.

The files with the template code you need are on the LEARN Ultra page for the assignment (click on "Assessment" on the LHS menu, then "Coursework 1 - Language Identification and Classification"). You will download a file called `assignment1.tar.gz` which can be unpacked using the following command to a shell prompt:

```
tar -xf assignment1.tar.gz
```

This will create a directory `assignment1` which contains additional Python modules used in this coursework, together with a file named `template.py`, which you must use as a starting point when attempting the questions for this assignment.

There is an interim checker that runs some automatic tests that you can use to get partial feedback on your solution: https://sweb.inf.ed.ac.uk/ititov/fnlp_interim_1.cgi

Submission

Before submitting your assignment:

- Ensure that your code works on DICE. Your modified `template.py` should fully execute using python3 with or without the answer flag.
- Test your code thoroughly. If your code crashes when run, you may be awarded a mark of 0 for the code-based questions and you will get no feedback other than "the code did not run".
- Ensure that you include comments in your code where appropriate. This makes it easier for the markers to understand what you have done and makes it more likely that partial marks can be awarded.
- Any word limits to open questions will be strictly enforced. We use the command `wc -w`, which is equivalent to `len(text.split())` in Python, to calculate the word count. If your answer exceeds the word limit, you may be awarded a mark of 0 and you will get no feedback other than "the answer exceeds the word limit".
- We may use automated tools to check your code and/or your text answers for evidence of academic misconduct.

The submission is divided into two parts: uploading the Python code via LEARN Ultra and answering the open questions (Question 1.3, Question 1.5, Question 1.7, Question 2.2 and Question 2.3) via Gradescope.

Python code submission - When you are ready to submit, rename your modified `template.py` with your matriculation number: e.g., `s1234567.py`. Submit this file by uploading it using the interface on the LEARN Ultra website for this piece of coursework. If you have trouble please refer to the blogpost here.

Answering the open questions - You have to answer the open questions (Question 1.3, Question 1.5, Question 1.7, Question 2.2 and Question 2.3) via Gradescope. You can access Gradescope through the assignment page on LEARN Ultra. When answering the questions on Gradescope, please refer to the question numbers in parentheses which reflect the question numbers in this document. Notice that there is a "Total word count" question at the end of each question. Please provide the combined word count across all subquestions in the form of an integer.

The deadline for submission is **12 noon GMT, Wednesday 14th February 2024**.

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you may not resubmit afterwards. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit exactly once after the deadline, and a late penalty will be applied to this submission, unless you have received an approved extension. Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same time frame as for on-time submissions.

For information about late penalties and extension requests, see the School web page here. Do not email any course staff directly about extension requests; you must follow the instructions on the web page.

Good Scholarly Practice: Please remember the School's requirements regarding all assessed work for credit. Details about this can be found at: `http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Furthermore, 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 (e.g., GitHub), then you must set access permissions appropriately (for this coursework, that means only you should be able to access it).

1 Language Identification

In this part of the assignment, we will build a character-level language model using the Brown corpus. To do this, we will use `LgramModel` from `nltk_model`, provided in `assignment1.tar.gz`. It is a small modification of the `NgramModel` you will use in Lab 1: `NgramModel` builds n-grams of words, while `LgramModel` builds n-grams of characters (letters).

In addition to the Brown corpus, we will use a small snapshot of data generated by users of Twitter in a form of tweets – messages that are at most 280 characters, shared publicly by the users of the platform. All of the tweets that are in the data set we use are from 28^{th} of January, 2010 (`20100128.txt` file in the `twitter` directory). Please note that the *order* of the tokens within each tweet has been scrambled—the tokens *themselves* have not been changed. We will apply the character-level language model trained on the Brown corpus to each word in a tweet in order to analyse whether it is written in English or not.

Corpus preparation: When preparing corpus data, whether from the NLTK Brown corpus or our twitter data, you should convert everything to lower-case. It is important to do this *after* you remove non-alphabetic tokens, as the conversion from upper- to lower-case is not well-defined for some characters in some writing systems.

Question 1.1 [7.5 marks]

Complete the function `train_LM`, which we use to train a character-level (or letter-level) language model. In this function, perform the following steps:

1. Create a list of all alphabetic tokens in a corpus (hint: use Python's `.isalpha()` string method.)
2. Train a bigram letter language model using the cleaned data from step 1 (hint: Look at the `LgramModel` code in `nltk_model/ngram.py`, particularly the `__init__` method). For this question, as in the 'Going Further' sections of lab1 and lab2, you should turn on both left and right padding. Do *not* supply your own estimator. `LgramModel` will supply a default smoothing estimator.
3. Return the trained `LgramModel`

Using this function, train a (lower-case, English) letter language model on the Brown corpus. As the Brown corpus is rather large, training the letter language model will take some time.

Question 1.2 [7.5 marks]

Clean up the Twitter corpus to remove all non-alphabetic tokens and any tweets with fewer than 5 tokens remaining (i.e. after token removal). Using the bigram letter language model that you trained in Question 1, complete the function `tweet_ent` to compute the average per-word letter entropy for each tweet in the 'cleaned' version of the Twitter corpus. The function should return a list of pairs of the form:

$$[(entropy1, tweet1), (entropy2, tweet2), ..., (entropyN, tweetN)]$$

where N is the number of tweets in the 'cleaned' version of the Twitter corpus. The list should be sorted in ascending order of average per-word entropy value. (hint: remember you have an `LgramModel` and tweets in the form of lists of words. You will need to compute the letter entropy of each token in the (cleaned) tweet, with left and right padding, and then normalise by "sentence" length. Be sure to review the arguments to `LgramModel.entropy`.)

Question 1.3 [3 marks]

Briefly explain what left and right padding accomplish and why they are a good idea. Assuming you have a letter bigram model trained on a large enough sample of English that all the relevant bigrams have reliable probability estimates, give an example of a string whose average letter entropy you would expect to be (correctly) greater with padding than without and explain why.

There is a 75-word limit for this question.

Question 1.4 [3 marks]

Perform the following experiment, starting from your `assignment1` directory:

```
python3 -i [your solution].py
>>> lm.entropy('bbq',verbose=True,perItem=True)
```

You should see something like this:

```
p(b|('<s>',)) = [2-gram] ...
p(b|('b',)) = [2-gram] ...
backing off for ('b', 'q')
p(q|()) = [1-gram] ...
p(q|('b',)) = [2-gram] ...
p(</s>|('q',)) = [2-gram] ...
...
```

If your output looks more or less like the above, copy-paste it into your code in the space for the answer to Question 1.4, otherwise copy-paste the above seven lines[1]. Then add a short comment to each line, explaining it.

There is a 75-word limit for the added comments.

Question 1.5 [3 marks]

Using the `hist` function supplied in `template.py`, plot histograms of the average letter entropy of the tweet data as returned by the `tweet_ent` function. The red line labelled '0' is the mean of the entropy distribution. The pink lines show multiples of the standard deviation.

Briefly describe the plots, what they tell you about the data, and how they suggest entropy might be useful in identifying different kinds of tweets.

There is a 75-word limit for this question.

Question 1.6 [10 marks]

Build a classifier to distinguish English tweets from non-English tweets, by implementing `is_English`, which should return True if the tweet (as pre-processed from the previous question) is classified as English and False otherwise. You can implement this function as you like. However, you must not use any data beyond the Brown and Twitter corpora we have given you, and no libraries beyond `nltk_model`, `numpy` and python's standard libraries[2] (e.g. `math` and `re` are fine). In particular, you are not allowed to use any NLTK libraries other than `nltk_model` as supplied.

We provide a small sample of annotated tweets on which you can test your implementation. You will be marked based on the performance of your classifier on the sample tweets as well as on a larger collection of unseen tweets. You may want to check the behaviour of your classifier on additional subsets of the (unannotated) Twitter corpus.

Hint: you are allowed to use the entropy from the letter language model that was trained on the Brown corpus, that is, the return value of Question 1.1. The return value of Question 1.1 is passed as the argument `bigram_model` into `is_English`.

Question 1.7 [16 marks]

> This is an **essay question** that is completely independent of the Twitter data and the Brown corpus and the preprocessing we asked you to do in the previous questions.

Suppose you are asked to find out what the per-word (*not* per-letter) entropy of English is.

[1] Taking this option will reduce your mark by 0.5

[2] https://docs.python.org/3/library/index.html

1. Name 3 problems that this question glosses over (in particular, any implicit assumptions that are problematic or require additional information to give a meaningful answer)[3].
2. What kind of experiment would you perform to estimate the per-word entropy? Be specific about the data and method you would use and justify the main design decisions you make in your experiment. Be sure to include the formula you would use.
3. Elaborate on the temporal dynamics of language evolution (i.e. etymology) and its implications for per-word entropy. How might the meaning and usage of words change over time[4], and how can your experimental design capture these changes?

There is a 600-word limit for this question.

2 Naive Bayes and Logistic Regression

In this part of the coursework, we look at classification, specifically at resolving attachment ambiguity of prepositional phrases (PPs). We only consider the ambiguity whether a given PP attaches to an NP in object position or to the VP:

Here is an example: the phrase "imposed a gradual ban on virtually all uses of asbestos" has two readings. Attachment to the NP:

$$[_{VP} \text{ imposed } [_{NP} \text{ a gradual ban } [_{PP} \text{ on virtually all uses of asbestos}]]]$$

and attachment of the PP to the VP:

$$[_{VP}[_{VP} \text{ imposed } [_{NP} \text{ a gradual ban}]] \, [_{PP} \text{ on virtually all uses of asbestos}]]$$

The data we are going to use is from Ratnaparkhi et al. (1994), who extracted those phrases from the Penn Tree Bank and removed all words except the "head words" in order to reduce data sparsity. The head words are the verb (imposed), the head of the object NP (ban), the preposition (on) and the head of NP embedded in the PP (uses). We use raw accuracy as our evaluation metric, i.e. the proportion of correctly resolved attachments out of all examples.

If you want to inspect the data directly, you can find the directory (with read access) on DICE here:

`/usr/share/nltk_data/corpora/ppattach/`

Question 2.1 [15 marks]

Implement a Naive Bayes classifier with Lidstone smoothing (as discussed in the lecture) **from scratch**, i.e. without using any of the functionality provided by NLTK. While the classification problem we consider only requires two classes, your implementation should also work for more classes.

Notation: we use the notation from the lecture: c represents a class, f represents a feature, F the set of all possible features in the training data and d ("document") an example that is annotated or that we want to classify.

Implement the following functions:

[3]Examples: Suppose someone asks: "How far do I get with a car with a full tank?" We cannot give a meaningful answer to this question directly because there are a lot of hidden assumptions. The model of the car, including the volume of the tank, (even the temperature has an effect on the volume of the fuel!), the passengers, the environment (traffic, how hilly is it?) all affect the answer. How exactly is "far" defined? As the crow flies or what the meter shows (relevant for hills, city driving, etc.)?

[4]Examples: Historically, the word "sick" maintained a negative meaning of physical or mental illness. In contemporary informal language, "sick" is often used with a positive connotation to express admiration or excitement. For instance, someone might describe a great performance, a stylish outfit, or an impressive accomplishment as "sick."

- `get_vocab` (1 mark) which takes the training data as a list of tuples of the form `(list with features, class-label)` and computes the set of all features used in the training data for all classes. Use this set for smoothing.
- `train` (8 marks) which takes the training data as a list of tuples of the form `(list with features, class-label)`, the α value for smoothing and the vocabulary of all possible features F. It computes probabilities for $P(c)$ and $P(f|c)$. Use Lidstone smoothing with α for $P(f|c)$ and MLE for the prior $P(c)$.
- TODO: consider asking the students to compute $P(c, d)$ first before the sub-question below.
- `prob_classify` (5 marks), which takes a list of features and returns the probability $P(c|d)$ for all classes as a dictionary. If d has features $f \notin F$, ignore those. $P(c|d)$ can not only tell us what the most likely class, it also tells us about the confidence of our model.
- `classify` (1 mark) which takes a list of features and computes the most likely class.

Question 2.2 [15 marks]

A friend of yours used a logistic regression model instead of Naive Bayes and computed the model's accuracy on the development set for different ways to extract features:

Features	**Example**	**Acc**
$[V]$	[imposed]	65.93
$[N_1]$	[ban]	66.20
$[P]$	[on]	74.13
$[N_2]$	[uses]	61.82
[“V” $= V$, “N_1” $= N_1$, “P” $= P$, “N_2” $= N_2$]	[V=imposed, N_1=ban, P=on, N_2=uses]	81.08

Table 1: Accuracy of a logistic regression model for different ways to extract features.

1. How do you interpret the differences in accuracy between the different ways to extract features? In particular, what does it say about the task? [7 marks]
2. In the template code for the previous question, we used the feature extractor in the last row. Compare the accuracy you got in the output for Question 2.1 with the Naive Bayes model to the results in table 1. If there is a difference, what might be the reason for that? [3 marks]
3. Your friend proposes to use a binary feature that is 1 if V=‘imposed’ AND N_1 = ‘ban’ AND P=‘on’ AND N_2 = ‘uses’. Otherwise, the feature takes the value 0. Would you advocate for or against using such a feature? Why? [5 marks]

There is a 150-word limit in total for this question (across all three sub-questions).

Question 2.3 [10 + 10 marks]

What is the best classifier we can get for disambiguating PP attachment? In this question, we want you to write your own feature templates for a Logistic Regression model.

Implement the function `your_feature_extractor`. You may use NLTK functions or write additional code as long as it does not require files or other dependencies that are not present on DICE. Note that training the model may take a while.

Inspect the features with the highest (absolute) weights (using the method `show_most_informative_features`).

1. Briefly describe your feature templates and your reasoning for them.
2. Pick three examples of informative features and discuss why they make sense or why they do not make sense and why you think the model relies on them. The examples you give should differ in an interesting way.

There is a 300-word limit for this question. The marks for this question are divided into 10 marks for the text and examples you provide and 10 marks for performance on the development set.

References

Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.