Coursework 1 – Foundations of Natural Language Processing

Deadline: 4pm, Monday 28th February 2022

1 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 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://homepages.inf.ed.ac.uk/cgi/ht/fnlp/interim1_22.py

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 and you'll 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 character limits to open questions will be strictly enforced. Answers will be passed through an automatic filter that only keeps the first N characters, where N is the character limit given in a question.

When you are ready to submit, rename your modified `template.py` with your matriculation number: e.g., `s1234567.py`.

Submit this file via LEARN by uploading it using the interface on the LEARN website for this piece of coursework. If you have trouble please refer to the blogpost here.

The deadline for submission is **4 PM, Monday 28th February 2022**.

Monday week 6

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 afterward. (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).

2 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 have seen before: in `NgramModel` we build 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). The order of the tokens has been scrambled. We will use the language model trained on the Brown corpus to analyse whether a tweet 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.

2.1 Question 1 [7 marks]

Complete the function `train_LM`, which we use to train a character-level (or letter) 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 the `nltk_model`, 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 letter language model on the Brown corpus. As the Brown corpus is rather large, training the letter language model will take some time.

2.2 Question 2 [7 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). Given the bigram letter language model that you trained in Question 1, complete the function `tweet_ent` in which you compute the average word 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 word entropy value. (hint: remember you have an `LgramModel` and tweets in a form of lists of words. You will need to compute the entropy at the word-level, with left and right padding, and then normalise by "sentence" length. Be sure to review the arguments to `LgramModel.entropy`.)

2.3 Question 3 [8 marks]

Inspect the list of entropy-tweet pairs generated in question 2. What differentiates the beginning and end of the list of tweets and their entropies? Complete function `open_question_3` with your answer. There is a 500 character limit on this question.

2.4 Question 4 [8 marks]

The Twitter data is still noisy, in that not all character sequences are legitimate spellings and/or words of English, despite having removed non-alphabetic strings. Look at the tweets and see what (if any) currently missing preprocessing steps would be likely to improve results. Describe in detail the problems that you have identified with the data and the techniques that you could use for the cleaning up process. Give examples where appropriate. There is a 500 character limit on this question; your answers do not need to be complete sentences and you are not required to implement your suggestions.
Hint: you may want to write the tweets before and after preprocessing into a file to make inspecting them easier.

2.5 Question 5 [15 marks]

Now we will do some tweet filtering to remove tweets that probably aren't written in English, based on their average word entropy. Complete function `tweet_filter` which performs the following steps:

1. Some of the tweets have non-ASCII characters, making them likely non-English. Create a list of ASCII tweets by removing the bottom 10% of tweets. To do this, use the list of average word entropy-tweet pairs, computed in the previous question. The resulting list should be sorted in ascending order of average word entropy value.

2. Using the ASCII tweets from the previous step, compute their average word entropy mean m and standard deviation σ (hint: consider using the numpy module for these computations.)

3. Using these statistics, compute a threshold $t = m + \sigma$ that allows us to filter out ASCII tweets that are further away from the mean than one standard deviation. Use t to obtain a list of non-English tweets that are ASCII tweets above this threshold. The resulting list should be sorted in ascending order of average entropy value.

Overall, `tweet_filter` should return mean and standard deviation of the ASCII tweets, and lists of ASCII and non-English tweets.

2.6 Question 6 [15 marks]

Suppose you are asked to find out what the average per word entropy of English is.

1. Name 3 problems with this question, and make a simplifying assumption for each of them.
2. What kind of experiment would you perform to estimate the entropy after you have these simplifying assumptions? Justify the main design decisions you make in your experiment.

This is an open question with no single correct answer. There is a limit of 1000 characters for this question.

3 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 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/
```

3.1 Question 7 [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:

- `get_vocab` (1 mark) which takes the training data as a list of tuples of the form (list with features, 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, 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 $P(c)$.

- `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.

- `classify` (1 mark) which takes a list of features and computes the most likely class.

3.2 Question 8 [10 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
$[\text{“}V\text{”} = V, \text{“}N_1\text{”} = N_1, \text{“}P\text{”} = P, \text{“}N_2\text{”} = N_2]$	[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.

- How do you interpret the differences in accuracy between the different ways to extract features? In particular, what does it say about the task?

- 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 7 with the Naive Bayes model to the results in table 1. If there is a difference, what might be the reason for that?

There is a 500 character limit for this question.

3.3 Question 9 [10 + 5 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`).

- Briefly describe your feature templates and your reasoning for them.

- Pick 3 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.

There is a 1000 character limit for this question. The marks for this question are divided into 10 marks for the text and examples you provide and 5 marks for performance on the development set (see table below).

Acc	**Marks**
< 81.20	0
≥ 81.20	1
≥ 82.15	2
≥ 83.10	3
≥ 84.05	4
≥ 85.00	5

Table 2: Marks out of 5 for accuracy on development set for question 9.

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*.