ANLP Assignment 2 2024

due: Wednesday, 20 November, 12pm (noon), via Gradescope

1 Overview of task, motivation, and goals

What will you do?

You will analyse and attempt to improve a simple model for **slot labeling**, a crucial component of **task-oriented dialogue systems** (like Siri and Alexa) that execute code in response to user commands. Specifically, you will do the following:

- analyse a tagger that uses word embeddings in a logistic regression model.
- analyse an algorithm that classifies spans using BIO tagging.
- analyse slot labeling output and attempt to improve it using linguistic features.

You will submit a written report of what you did and why. We will ask you to answer specific questions.

Why?

The most effective way to develop an NLP system is iterative: start with a simple solution, analyse its failures, attempt to improve it, and repeat. This assignment will give you experience in iterative development. In completing it, you will practice skills that are important for NLP and other areas of data science, by demonstrating that you:

- understand concepts of word embeddings, logistic regression, tagging, and span labeling.
- understand linguistic annotations like part of speech and dependency labels.
- understand how to interpret common NLP metrics.
- can apply these concepts to a new NLP problem that we haven't previously covered in class.

As with the first assignment, we expect for you to further demonstrate that you:

- can implement simple NLP systems in Python.

- can produce good scientific writing. This means clearly justifying your decisions, analysing your results, and drawing appropriate conclusions from them. We don't just care that you got the right answer, we also care that you are able to explain and justify your answer.

The only artifact that we will assess is your written submission, which must demonstrate your reasoning and explanatory skills. A simple implementation and analysis that is clearly reasoned and explained well will receive higher marks than a complex implementation and analysis that is poorly reasoned or badly explained, even if the complex implementation gets better results.

2 Working with others (and not)

You are strongly encouraged to work with a partner. You will learn more that way, since you will need to explain your reasoning to them, understand their perspective on the problem (which might be different from yours), agree a course of action, and decide how to share the work. These are all critical skills for working in most NLP and data science jobs.

You cannot work with the same partner who you worked with on assignment 1.

All other expectations around working together remain identical to those discussed in the handout for Assignment 1. Please review them, since we expect you to be familiar with them.

3 Submitting your assignment

- **Answer templates.** To prepare your submission, you must use either the .tex or .docx template linked here, in which you will fill in your answers to each question. Each answer should start on a new page and strictly observe the page limits for the question, or you will lose marks. This does not mean that answers must use all available pages; some answers can and should be shorter.
- **How to submit.** You will submit your assignment through Gradescope, via Learn. You should submit a single .pdf file, and if you are working in a group, **only one of you should submit a file**. You can add your partner to the submission in gradescope.
- **Late submissions.** As with Asgn1, no extensions are permitted, and late submissions will receive a mark of zero.

4 Assignment specification

4.1 Getting the Code and Data

The data for this assignment is from a dataset called NLU++ [Casanueva et al., 2022]. You can obtain both the code and the data from:
`https://git.ecdf.ed.ac.uk/anlp/course_materials/-/raw/main/current/assignments/assignment2/anlp_asgn2.zip`

Download and unzip the file. It will create a directory `anlp_asgn2` containing:

- `label_slots.py`: A runnable program with some baseline models to get you started. You will need to add some code to this file to complete the assignment.
- `utils.py`: Some helper functions to tokenise, annotate, and transform data; and to evaluate results. You do not need to modify any of the code in this file.
- `data/training_data.json`: Training data for your models.
- `data/validation_data.json`: Validation data for your experiments.
- `data/test_data.json`: Test data for reporting your final results.
- `data/ontology.json`: A file describing the possible intents and slots (see explanation below).

Data. The data is formatted in JSON, a standard text-based format to store structured data. You can view the data files at the command line or in most browsers. The Python standard library includes functions to read and write JSON, so you do not need to implement this yourself.

Code. We've provided code for some baseline models that you will need to analyse. You can complete the entire assignment by writing only a small amount of code. However, you do not need to submit any code, since you are being assessed only on your writing. You should not use any outside code or libraries, other than the libraries that are already included: `spacy` and `sklearn`.

4.2 Setting up your environment

Before you can run the code for this assignment, you will need to install some packages and data. We recommend that you do this using `conda`, which is installed on DICE. First, create a fresh environment to work in:

```
> conda create -n anlp_hw2
```

On some DICE machines, running `conda` may produce an error message like this:

```
Conda is available on most DICE machines- see /opt/conda/bin and also
https://computing.help.inf.ed.ac.uk/python for more information
```

If (and only if) this happens to you, try running:

```
> .  /opt/conda/bin/activate
```

Note the leading `.`!

You must do this before running `conda` on any machine that produces the above message.

Every time you log in to work on the assignment, you must to activate this environment:

```
> conda activate anlp_hw2
```

The first time you activate your environment, you should install the Python packages `spacy` and `scikit-learn`. This will make them available each time you activate the `anlp_hw2` environment.

```
> conda install -c conda-forge spacy
> conda install scikit-learn
```

You must also install a `spacy` model that the code needs. You will only need to do this once.

```
> python -m spacy download en_core_web_sm
```

Each of these commands may take a few minutes to run, but you only need to run them once. After that, you can always access them by activating your environment:

```
> conda activate anlp_hw2
```

You can deactivate this environment at any time (for example if you have a different environment for another course) by running:

```
> conda deactivate
```

4.3 Understanding the Problem and the Data

Before you get started, you should (re)read the following sections of JM3:

- Section 17.3, which discusses BIO tagging.
- Section 4.7, which discusses evaluation metrics.

The rest of the assignment assumes that you are familiar with these concepts.

For this assignment you will build NLP systems for **slot labeling**, a crucial component of dialogue systems. Your dialogue system helps hotel guests book rooms and services, and users interact with it via a messaging app. For example, a user might send a message like this:

```
I want to book 2 rooms from 4th to 8th of Jun.  It'll be for 1 adult in each room.
```

To respond to a request like this, a task-oriented dialogue system must solve three problems:

Problem 1: Intent classification. The system must first determine the **intent** of the user. In this case, the user wants to book a room, and the corresponding intent is called `booking`. We can model this as a text classification problem, and it is called intent classification. Its purpose is to classify each each input according to a pre-defined set of intents that the system can act on. We won't focus on intent classification in this assignment, but your data includes the intents so that you can see what they look like. (Notice that texts can have multiple intents or zero intents!)

Problem 2: Slot labeling (The focus of this assignment). Imagine that your system has correctly identified the `booking` intent from the above request. To help the user, the system must search for available rooms that meet the user's needs by querying a database. The query must supply certain named pieces of information:

- DATE_FROM: When should the booking start?

- DATE_TO: When should the booking end?
- ADULTS: How many adult guests will there be?
- KIDS: How many child guests will there be?
- ROOMS: How many rooms are needed?

Each of these named pieces of information is called a **slot**. The goal of **slot labeling** is to identify which parts of text are related to each slot. If some slots are missing, then the system can prompt the user for more information. (Many examples in the data are responses to such a prompt.)

Returning to our example, you will notice that the user has already supplied information about most of the slots. So, your system should label the relevant spans like this:

```
I want to book    2    rooms from    4th    to 8th of Jun .
                ROOMS            DATE_FROM     DATE_TO
It'll be for 1 adult in each room .
             ADULTS
```

Problem 3: Slot filling. The user input is free text, but database queries require canonical values. So, slots are filled by converting the labeled text to canonical values. For example, we need to convert `8th of June` to `08/06/2022` and convert `1 adult in each room` to `2`. If you suspect that this is a difficult problem, you are right! Intent classification, slot labeling, and slot filling are *all* difficult problems. For this assignment, you will only work on slot labeling. But understanding this complete pipeline will help you appreciate the complexity of building a real NLP system.

The data we have given you is annotated for all three tasks described above. However, the assignment focuses only on slot labeling. You can limit your focus to the following attributes for each example in your data:

- `text`: an input text.
- `slots`: a set of slot labels that apply to the text.
- `span`: the span associated with a slot.

Look at your training data to see some examples.

```
> less data/training_data.json
```

4.4 Running the Baseline Code

Slot labeling is a **span classification** problem, which can in turn be modeled as a tagging problem by using a **BIO tagging** scheme. In a BIO tagging scheme, the first word of every labeled span is tagged with a label that starts with `B-` (*begins*) while the remaining words are tagged with a label that starts with `I-` (*inside*). Words that are not part of a span are tagged with `O` (*outside*).

(Using `B-` and `I-` tags enables your system to distinguish between cases where a single label covers a span of words, and cases where multiple labels cover that span.) So our running example would be tagged like this, after being tokenised (which is handled for you):

```
I  want  to  book     2     rooms  from      4th      to     8th        of        Jun     .
O   O    O    O    B-rooms    O     O    B-date_from   O   B-date_to  I-date_to  I-date_to

           It  'll  be  for     1       adult       in       each      room    .
            O   O   O   O   B-adults  I-adults  I-adults  I-adults  I-adults  O
```

We have given you a simple baseline model that tags words according to their *most frequent tag* in the training data. In the code, you will see that it is described as a model of $P(\langle \text{tag} \rangle \mid \langle \text{word} \rangle)$. More precisely, given a sequence $w = w_1 ... w_{|w|}$ of word tokens, where $|w|$ is the sequence length and w_i is the ith word, it models the conditional probability of a tag sequence $t = t_1 ... t_{|w|}$ as $P(t \mid w) = \prod_{i=1}^{|w|} P(t_i \mid w_i)$, where each $P(t_i \mid w_i)$ is estimated via (unsmoothed) maximum likelihood. It then simply chooses $\text{argmax}_t P(t \mid w)$. This is a common baseline for tagging models, though as we shall soon see, it is a poor baseline for a BIO tagging model.

To train the baseline model, apply it to the validation data, and evaluate the results, run:

```
> python label_slots.py
```

The program will take several seconds to run; it will print results and exit. You should see BIO tagging results, reported in terms of *precision*, *recall*, and *f1-score* for each label, along with the *support* (number of examples) for each label type. At the end, you will see *micro-averaged* and *macro-averaged* f1 scores. There is also a weighted average, which is a compromise between micro- and macro-averages.

You will notice an error message at the end of the report. We'll return to it in Section 4.5.2.

The baseline script has several options to change its behaviour. To see them, run:

```
> python label_slots.py -h
```

For example, to see a detailed report of how the model tagged each word, run:

```
> python label_slots.py -f
```

Take a few minutes to familiarize yourself with this report. It shows BIO tags and the corresponding spans, along with some linguistic annotations of the data obtained with `spacy`.

4.5 Tasks

4.5.1 Logistic regression tagging with word embeddings (20 marks)

You may have noticed a problem with the most frequent tag model: it cannot label any words that it hasn't seen before. To circumvent this problem, we can condition the tag on something that is shared across words, including unknown words. The code includes a *logistic regression* model that conditions the choice of tag on a word's *embedding*, rather than the word itself. Since word embeddings for semantically similar words tend to be similar, conditioning on the word embedding

might enable the model to correctly predict tags for words that were not in the training data. This is still a model of $P(t \mid w) = \prod_{i=1}^{|w|} P(t_i \mid w_i)$, but $P(t_i \mid w_i)$ is now a logistic regression model. Hence it is a drop-in replacement for the baseline model. Use this command to run it:

```
> python label_slots.py -m logistic_regression
```

Compare the results of the most frequent tag and logistic regression models by looking at their reports. Identify a BIO tag type on which one system improves over the other, and examine the output of both systems on this tag type by using the full report. Explain why you think one system is better on this tag type, and justify your answer using examples.

Your answer to this question must not exceed one page. A good answer can be considerably shorter than this, perhaps one or two paragraphs.

4.5.2 BIO tagging for slot labeling (30 marks)

You will have noticed this message at the end of the report:

```
!!! Classification report for slots is unavailable because some predicted BIO tag sequences were malformed !!!
```

For a BIO tag sequence to be correctly formed, every `I-` tag must follow a `B-` or `I-` tag of the same type. Convince yourself that this is true. The code contains an algorithm that can enforce this constraint. Use this command to run it with the logistic regression model:

```
> python label_slots.py -m logistic_regression -p bio_tags
```

The error message should go away, and you should see a classification result for slot labels.

The algorithm that enforces these constraints is in the function `predict_bio_tags`. Compare it to the baseline algorithm `predict_independent_tags`. You will notice that it still uses the same model, $P(t \mid w) = \prod_{i=1}^{|w|} P(t_i \mid w_i)$, to assign probability to possible sequences t. But it will only choose a t that obeys the constraints above. Once you have examined the algorithm, answer the following questions in your submission: What does the algorithm do, and why does it work? How would you adapt another algorithm that you saw in class to choose a tag sequence that obeys these constraints, without changing the model or its estimation procedure? (You do not need to implement the other algorithm; only to name it and describe any necessary adaptations.) Compare and contrast the algorithm in the code, and the algorithm you described: what are the benefits and drawbacks of each? Explain and justify your answers as precisely as you can.

Your answer to this question must not exceed one page.

4.5.3 Error Analysis and Feature Engineering (40 marks)

Now that you have seen different models and thought about their evaluation and analysis, your task is to attempt to improve the system even further. Analyse the results of the current system and find a span label that is frequently mispredicted (i.e. has high support along with low precision, low recall, or both). Looking at the true and predicted examples of this label, come up with a hypothesis about why the errors occur, and identify any provided linguistic features that might help your classifier avoid these errors. You may use any of the features provided by the spacy `en_core_web_sm` model, including word embeddings and other features that do not appear in the full report. You may use

features of the word itself or its neighboring words, including the word embeddings. (Be careful with sentence boundaries; do not assume that adjacent sentences in the dataset are sequential turns from the same dialogue.) For this question, you may not use features from other software or models. Once you have identified a candidate feature (or features), implement `train_my_model` to use both the word's embedding and the feature (or features) that you have identified, and run it:

```
> python label_slots.py -m my_model -p bio_tags
```

Your answer to this question in your submission should summarize your hypothesis, using examples that illustrate the problem and highlight the potential of linguistic features to help solve the problem. It should also explain whether you think your new model solved the problem as intended. Justify your answer using quantitative and qualitative evidence (i.e. numbers and examples). Your answer should consider the effect of your new model on your hypothesised problem, and on the overall performance of the system.

Your answer to this question must not exceed two pages. A good answer can be considerably shorter than this.

Your mark for this question will not depend on your evaluation results. A well-motivated, simple model with a careful analysis of the results will receive high marks, even if your results do not improve over the baseline, and even if they are worse! A poorly-motivated, complex implementation with little analysis will receive low marks, even if the results are state of the art. The purpose of this question is for you to demonstrate your critical thinking and analytical skills, and your engagement with the concepts in the course, not your machine learning prowess.

4.5.4 Final Test and Reflection (10 marks)

Until this point, we have asked you to evaluate all systems on the *validation* data. We have not asked you to report anything on the *test set*. This choice is deliberate. You need a dataset to evaluate your incremental progress on a system (including debugging your code). However, once you have run an algorithm on a dataset and looked at the results, it is no longer a true test set. Every further decision that you make may be influenced by your observations of the data, whether you intend this or not. This is why we have asked you to only evaluate on the validation data so far, and not the (final) test data. For more information about this methodological perspective, see `https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets`.

Now it is time for the final test! Evaluate two models on the test set. For the first, run:

```
> python label_slots.py -m logistic_regression -p bio_tags -v data/test_data.json
```

For the second, run:

```
> python label_slots.py -m my_model -p bio_tags -v data/test_data.json
```

Your answer to this question should compare and contrast the results of the two models on the validation and test sets. Are the test results as you expected? Why or why not? Justify any claims in your answer using quantitative and qualitative evidence (i.e. numbers and examples). Strong answers to this question will include claims backed by evidence rather than speculation.

Your answer to this question must not exceed one page. A good answer can be considerably shorter than this, perhaps one or two paragraphs.

5 Marking and how to do well

The marking scale and expectations for your submission remain identical to those described in Section 5 of Assignment 1.

References

Inigo Casanueva, Ivan Vulić, Georgios Spithourakis, and Paweł Budzianowski. NLU++: A multi-label, slot-rich, generalisable dataset for natural language understanding in task-oriented dialogue. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1998–2013, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.154. URL https://aclanthology.org/2022.findings-naacl.154. Original data available at https://github.com/PolyAI-LDN/task-specific-datasets/tree/master/nlupp.