Lab 3 - Deadline: February 27

Introduction

This lab will cover necessary topics more in-depth (based on received feedback), where the majority of work will be placed in the exercise questions (part 1). In contrast, the implementation (part 2) will be much more open to interpretation and experimentation. Please contact me or use the forum if you struggle with the lab exercises.

We have seen how you can use the NLTK and spaCy libraries for various tasks. You should by now have a fair understanding of word vectorisation (e.g. with TF-IDF) and how POS tags come in handy for processing text in general. The tasks in this lab will focus on these topics (details will be in their respective sections):

1. POS chunking
2. Dependency parsing
3. External knowledge using Wordnet and Sentiwordnet
4. Building a sentiment analysis model

The spaCy youtube channel has a good video introducing the usage of POS chunks and dependencies: https://www.youtube.com/watch?v=BoyLPiXXEYA

Setup

You will be using the following libraries:

- NLTK (POS chunking, Wordnet, Sentiwordnet)
- spaCy (dependency parsing)
- scikit-learn (training a classifier) [**NEW**]

```
pip install -U scikit-learn
```

For usage of scikit-learn, see these examples:

- https://scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_text_feature_extraction.html

Part 1: Exercise questions

1. POS chunking

Chunking allows us to group words based on their POS tags. This is useful for finding meaningful groups of words, such as noun phrases (NP) or verb phrases (VP). Given the sentence “The quick brown fox jumps over the lazy dog”, we have the following POS tags (using the universal tagset):

```
('The', 'DET') ('quick', 'ADJ') ('brown', 'ADJ') ('fox', 'NOUN')
('jumped', 'VERB') ('over', 'ADP') ('the', 'DET') ('lazy', 'ADJ') ('dog', 'NOUN')
```

Answer the following (using any tools available from NLTK or spaCy):

1. Create a chunker that detects noun-phrases (NPs) and lists the NPs in the sentence.
    - Both NLTK and spaCy supports chunking
2. Modify the chunker to handle verb-phases (VPs) as well.
3. Can chunking be beneficial in the context of a predictive keyboard? Why or why not?

2. Dependency parsing

Dependency parsing can be used to extract the grammatical structure of a sentence. A particularly useful feature is to fetch the subject/objects and the root of a sentence.

Use spaCy to inspect/visualise the dependency tree of the sentence "The quick brown fox jumps over the lazy dog" and answer:

1. What is the root of the sentence?
2. Write a function to get the subject and object of a sentence. Print the results for the sentence above.
3. How could you use the relationships from dependency parsing in a predictive keyboard?

3. External knowledge using Wordnet and Sentiwordnet

These are both lexical resources that can be used to extract more information from a word than what is available in the text itself. Wordnet is a lexical database for the English language, which can be used to find synonyms, definitions and more. Sentiwordnet is a lexical database for words and their associated opinionated values.

Early sentiment models relied on these rule-based approaches, but we have since moved on to machine learning. However, while simplistic, they can still prove to be valuable. A singular word can sometimes be considered strictly positive or negative for next-word prediction.

Solve the following tasks:

1. Use Wordnet (from NLTK) and create a function to get all synonyms of a word.
2. Explore other features you can extract from Wordnet for the same word. Are there any features that could be useful for a predictive keyboard?
3. Use Sentiwordnet (from NLTK) and create a function to get the sentence's sentiment: "Well, I don't hate it, but it's not the greatest!"
   - Consider features from Wordnet to get a more reliable sentiment score.
   - Can you think of a way to alter texts to handle negations? (such as "not the greatest")
   - Explain your approach

4. Building a sentiment analysis model

- The model should be able to predict a text's sentiment, whether that is binary (negative/positive) or on a scale (also called fine-grained).
- You can select between two data sources based on your preferences.
  - [SIMPLE] Twitter samples. ~20k tweets (binary classification)
    - https://www.nltk.org/howto/twitter.html#Using-a-Tweet-Corpus
  - [ADVANCED] Amazon Reviews with 1-5 star ratings (fine-grained).
    - A subset of the very large Amazon review corpus (total 55 GB). The provided data is 50 MB (96834 reviews), and only considers the "major appliances" category.
    - The file can be found under `data/amazon_reviews`. Make sure to update your repository!
    - Feel free to download more data and change up the processing pipeline to suit your needs (e.g. if you wish to create a language model based on review data)
      - https://www.kaggle.com/datasets/cynthiarempel/amazon-us-customer-reviews-dataset

Requirements:

- The model should be trained on a training set and evaluated on a test set.
- The evaluation score should be reported and preferably have an F1-score above 0.7.
- The data should be passed through a typical pipeline: cleaning/normalisation, vectorisation/feature extraction, training and evaluation.

- The model should be developed using scikit-learn

All other details are up to you to implement. Whether you select a Naive Bayes model or move on to more advanced approaches with hyperparameter tuning of ensemble methods is up to you.

Tips:

- You can print out evaluations using `sklearn.metrics.classification_report`.
- NLTKs TweetTokenizer might come in handy.
- You can use the sentiwordnet-approach you developed in task 3 to define a baseline.

Part 2: Implementation

In Lab 2, you implemented part-of-speech to some extent (whether that was only producing a specific POS tag or filtering). In this lab, you can perform a similar process based on results from the sentiment analysis model(s) built in part 1 of this lab. There are various ways to approach this task! Examples could be a preset sentiment definition (angry or happy) or a more dynamic approach based on the text you are currently typing. You can also use POS-tagging to have a better selection of candidate words for sentiment analysis (e.g. by looking only at verbs).

Additionally, multi-word prediction is an interesting topic for a predictive keyboard. This could be implemented by using frequent occurrences of specific part-of-speech chunks (e.g. noun-phrases) and treating these as a single token. This is an optional task. If you want to experiment with other type of language models (based on what you've now learned), feel free to do so! Please come to the lab sessions or office hours to discuss your ideas.

The basics are, as always, prepared in `lab3.py` under `your_implementations`.

Requirements for part 2

- A working Lab3 class with `train` and `predict` methods
- A next-word prediction language model (based on previous labs or a new implementation if you wish)
- Integration of a sentiment model (either lexicon-based or a machine learning model) to alter or filter predictions

Recommended/optional

- Multi-word prediction (e.g. by clever usage of POS-chunking or dependency parsing)
- A more sophisticated language model (if you wish to experiment with neural networks, that's fine too!)

Lab notes and feedback

As a final part of the lab, I want you to briefly discuss your approach to the implementation task. This could be things you had to learn, difficult parts of the lab, etc.