Artificial Intelligence

Tutorial week 8 – Natural Language Processing

COMP9414

Theoretical Background

Sentiment Analysis is the task of determining the emotional tone behind a certain text. Sentiment can be classified as either *positive*, *negative* or *neutral* depending on the attitude expressed in the text. The following texts provide examples of positive, negative and neutral sentiment:

- "This is beautiful! You really did a great job" – Positive
- "Whaat? This is all wrong! I'm not happy with this at all" – Negative
- "I guess it could be worse" – Neutral
- "Oh yeah, great job breaking my only laptop!" – Negative

The last example exhibits a peculiar phenomenon in sentiment analysis called *sarcasm detection* (i.e. sentences that *sound* positive but are really just being sarcastic and conveying a negative sentiment).

In this tutorial, we will use a supervised machine learning technique to create an automated sentiment classifier for movie reviews. This will also give us a chance to familiarise with popular NLP techniques (such as Regular Expressions and Context-free Grammars as well as libraries that researchers in the field use to make their job easier (such as NLTK and Scikit-Learn).

1 Grammars

A **formal grammar** is a set of rules that define how to generate and recognise strings that belong to a certain Language L. Grammars are designed to formalise the *syntax* of a language (i.e. how to write things that make sense in that language) and give no information on its *semantics* (i.e. what sentences in that language mean).

We can formally define a grammar as follows:

$$G = (V, \Sigma, R, S). \tag{1}$$

Where:

- V is a finite set of *non-terminal symbols* or *variables*. These are the symbols used in the grammar to denote syntactic categories.
- Σ is a finite set of *terminal symbols* which make up the actual content of the generated sentences. This is also known as the *alphabet* of the language generated by G.
- R is a set of relations defined in $V \times (V \cup \Sigma)^*$. These are also known as *Rewrite Rules* and indicate how grammar variables can be converted into other variables or into terminal symbols.
- S is the *Start Symbol* used to represent the whole sentence.

In this section we will explore some basic generation rules for grammars, using them as a tool to generate training data for our Sentiment Classifier

1.1 Grammar Exercises

a. Write a grammar `greetGrammar` that generates the following three strings: `["hello", "hi", "good to see you"]`

b. Using NLTK, load your grammar and verify that the strings it generates the correct language

c. Modify the `greetGrammar` to accept a name after the greeting. The new grammar should parse all greet strings followed by either of these names: `["Alice", "Bob", "Charlie"]`. Test your grammar with `NLTK` to verify that it produces the correct language
(**Hint:** use a new non-terminal symbol N to indicate the names and combine it with the existing rules)

d. We will now use grammars to automatically generate some data for our Sentiment Analysis classifier. Consider the following grammar that generates positive reviews for films:

```
S -> NP VP | PR VPR
NP -> Det N
N -> 'director' | 'screenplay' | 'plot' | 'story' | 'scenes' |
 'special effects' | 'costumes' | 'actors' | 'dialogues' | 'characters'

VP -> Verb Adj

VPR -> VerbPR NP

Det -> 'the' | 'this' | 'these' | 'those'

Verb -> 'is' | 'looks' | 'was' | 'are' | 'look' | 'were'

VerbPR -> 'love' | 'loved' | 'enjoy' | 'enjoyed'
| 'fell in love with' | 'adore' | 'adored'

PR -> 'I'

Adj -> 'great' | 'cool' | 'amazing' | 'fantastic' | 'very nice'
```

Load the grammar in NLTK and store all generated strings in a variable. Then, print 5 randomly generated strings from the grammar

e. You may have noticed that your grammar generates strings that are not consistent with singular/plural and with first/third person verbs. Fix the grammar so that it only generates strings that are grammatically correct.

f. Modify the `positive_film_grammar` to obtain a `negative_film_grammar` that produces negative reviews.

g. Generate a training dataset for the Movie Reviews Sentiment Analysis task. Your dataset should have 1,000 positive reviews obtained by sampling three random utterances from the `positive_reviews` language and concatenating them together, and 1,000 negative reviews obtained by applying the same method to the `negative_reviews` language. Each datapoint should be a tuple `T= (utterance, label)` , where `label` can be either "neg" or "pos" depending on the sentiment of the generated datapoint.

2 Data Preparation

A crucial component of every Natural Language Processing application is the *Data Preparation Pipeline*, which converts the data into a format that can be parsed by a statistical machine learning algorithm. This pipeline usually combines at least the following essential steps:

- **Data cleanup** (i.e. removing noisy elements from a dataset such as special characters, abbreviations, URLs, multiple spaces etc.)
- **Word Embedding** (i.e. converting text data into numerical vectors). This can be performed in a number of different ways, but usually involves the creation of a *Vocabulary*, which maps words to unique numerical IDs.
- **Data splitting and collation**, which involves dividing a dataset into different subsets for training, testing and validation of hyperparameters (splitting), as well as combining multiple datapoints into a dense batch for trianing (collation)

We will use NLTK's **Movie Review Corpus** as a dataset for our sentiment classifier. This is a well-known corpus for the task which can be downloaded and imported directly from a Python script using the `NLTK` library (note that we won't tokenize reviews at this stage, as tokenization will be handled later when building the Machine Learning model). We will use *Regular Expressions* for the Data Cleanup step, relying on Python's internal Regular Expression library, `regex`. For the other steps, we will rely on `scikit-learn`, a machine learning library highly specialised for textual data processing

2.1 Regular Expressions

A regular expression (often shortened as regex) is a sequence of characters that specifies a match pattern in text. This sequence is used in combination with a matching algorithm that finds the pattern in a text, possibly replacing it with another piece of text.

There are various implementations of regular expressions, including online portals, UNIX shell commands and libraries for basically every imperative programming language. In this tutorial, we will use Python's `regex` module and its `sub` function, which finds regular expression patterns and replaces every occurrence with a new piece of text.

a. Import the `regex` library as `re`, and use its `re.sub` function to replace all occurrences of the word "men" with the word "people" in the following portion of text (do not replace the string if it is a substring of another word):
```
"We hold these truths to be self-evident, that all men are
created equal, that all men are endowed by their Creator with
certain unalienable Rights, that among these are Life, Liberty
and the pursuit of Happiness and mental stability."
```

b. Load the NLTK Movie Reviews corpus. You will have to download the corpus first with the command `nltk.download("movie_reviews")` and then you will be able to load it by importing `movie_reviews` from `nltk.corpus`. Load the corpus utterances and labels in a list of tuples similarly to the dataset created in Section 1.g

c. Using regular expressions, write a function called `cleanup_review` which performs the following cleanup operations on your data:

- Replace all URLs with `URLTOKEN`
- Replace all dates with `DATETOKEN`
- Remove all non-alphanumerical characters (except for the following: `["?", "!", ",", ".", ":", ";", "'", "¨]`)
- Collapse multiple spaces into one space

Test your function with the following string:

```
"Hello!!  My name is Stefano, I have been a tutor for COMP#9414
since 01/04/2023.  My personal website is http://stefano.com .
(Nice to meet you ^__^)"
```

should return

```
"Hello!!  My name is Stefano, I have been a tutor for COMP9414
since DATETOKEN. My personal website is URLTOKEN . Nice to meet
you"
```

2.2 Data Splitting and Dataset Creation

a. Apply the `cleanup_review` function to the NLTK dataset. Then, split it into three subsets:

- `train_nltk_data` (which should contain the first 85% of the reviews (1-1700))
- `test_nltk_data` (which should contain reviews 85%-95% (1701-1900))
- `valid_nltk_data` (which should contain the remaining reviews (1901-2000))

Remember to shuffle the cleaned data before splitting to ensure an equal distribution of labels across the three sets. You can set a seed of 999 to ensure a replicable behaviour for your Machine Learning algorithm.

3 Machine Learning

In this last section, we will implement a statistical machine learning model to fit the dataset and correctly predict the sentiment on new reviews. We will use `scikit-learn` for this section, and experiment with different classes of ML models.

Scikit-learn offers a data structure called the `Pipeline` which allows to combine data transformation functions and machine learning algorithms into a single object. We will rely on that for this section, and combine a Support Vector Classifier with a TF-IDF Vectorizer. The former is a type of machine learning classification algorithm that is known to work particularly well with text classification tasks such as Sentiment Anaylsis, while the latter is a type of vectorizer that converts each word in a document to a numerical ID and weighs it according to how often it appears in the review and in the rest of the corpus; the algorithm prioritizes words that are popular across all reviews, and gives less priority to words that are specific to a single review but will not help the model to generalize to unseen ones.

3.1 Exercises

a. Create a `scikit-learn` pipeline comprised of two separate components:

   - A `TfidfVectorizer` with `min_df=3` and `max_df=0.95` called "vect"
   - A `LinearSVC` classifier with `C=1000` called "clf"

b. Train your pipeline with the 'fit' method, giving it a list of training samples and a list of labels associated with those samples. Use your `grammar_training_dataset` for this step

c. Run your pipeline on the `test_nltk_data`. Then, evaluate your pipeline by using the `classification_report` function from `sklearn.metrics`. Your performances will likely not be very good, due to the repetitive nature of data generated via grammars.

d. Train your pipeline again, this time using your `train_nltk_data` dataset.

e. Evaluate the performances of your model using `classification_report`. The new model should be a lot more accurate than the previous one.

f. Play with your code to familiarise with scikit-learn's suite of classifiers and parameters. Some experiments you may want to run include:

   - Training a model combining the `grammar_training_dataset` and `train_nltk_data` and verify whether it performs better than the `train_data` only model.

- Experiment with different classifiers – for example, you may want to try a simple `GaussianNB` classifier, or try some classifiers that usually perform well on this task such as `AdaBoostClassifier` or `RandomForestClassifier`.
- Try changing the parameters of your classifiers – for example, try reducing the `C` regularization parameter in your SVC, or increasing it further.