GENERAL ASSEMBLY

NATURAL LANGUAGE PROCESSING AND TEXT CLASSIFICATION

LEARNING OBJECTIVES

- Define natural language processing
- List common tasks associated with
  - use-cases
  - tokenization
  - tagging
  - parsing
- Demonstrate how to classify text or documents using scikit-learn

INTRODUCTION

NATURAL LANGUAGE PROCESSING

WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

- Natural language processing is the task of extracting meaning and information from text documents.

- There are many types of information we might want to extract.

- These tasks may range from simple classification tasks, such as deciding what category a piece of text falls into, to more complex tasks like translating or summarizing text.

- For most tasks, a fair amount of pre-processing is required to make the text digestible for our algorithms. We typically need to *add structure* to our *unstructured data*.

WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

- Many AI assistant systems are typically powered by fairly advanced NLP engines.

- A system like Siri uses voice-to-transcription to record a command and then various NLP algorithms to identify the question asked and possible answers.

TOKENIZATION

- Tokenization is the task of separating a sentence into its constituent parts, or *tokens*.

- Determining the "words" of a sentence seems easy but can quickly become complicated with unusual punctuation (common in social media) or different language conventions.

TOKENIZATION

- What sort of difficulties can you find in the following sentence?
- The L.A. Lakers won the NBA championship in 2010, defeating the Boston Celtics.

TOKENIZATION EXAMPLES

My house is located in Uptown. → [My, house, is, located, in, Uptown]

The Lakers are my favorite team. → [The, Lakers, are, my, favorite, team]

Data Science is the future! → [Data, Science, is, the, future]

GA has many locations. → [GA, has, many, locations.]

LEMMATIZATION AND STEMMING

- How would you describe the relationship between the terms ‘bad’ and ‘badly’ or ‘different’ and ‘differences’?
- *Stemming* and *lemmatization* help identify common roots of words.
- *Stemming* is a crude process of removing common endings from sentences, such as ‘s’, ‘es’, ‘ly’, ‘ing’, and ‘ed’.

LEMMATIZATION AND STEMMING

- Lemmatization is a more refined process that uses specific language and grammar rules to derive the root of a word.

- This is useful for words that do not share an obvious root such as ‘better’ and ‘best’.

- What are some other examples of words that do not share an obvious root?

LEMMATIZATION AND STEMMING EXAMPLES

Lemmatization	Stemming
shouted → shout	badly → bad
best → good	computing → comput
better → good	computed → comput
good → good	wipes → wip
wiping → wipe	wiped → wip
hidden → hide	wiping → wip

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. What other words or phrases might cause problems with stemming? Why?
2. What other words or phrases might cause problems with lemmatization? Why?

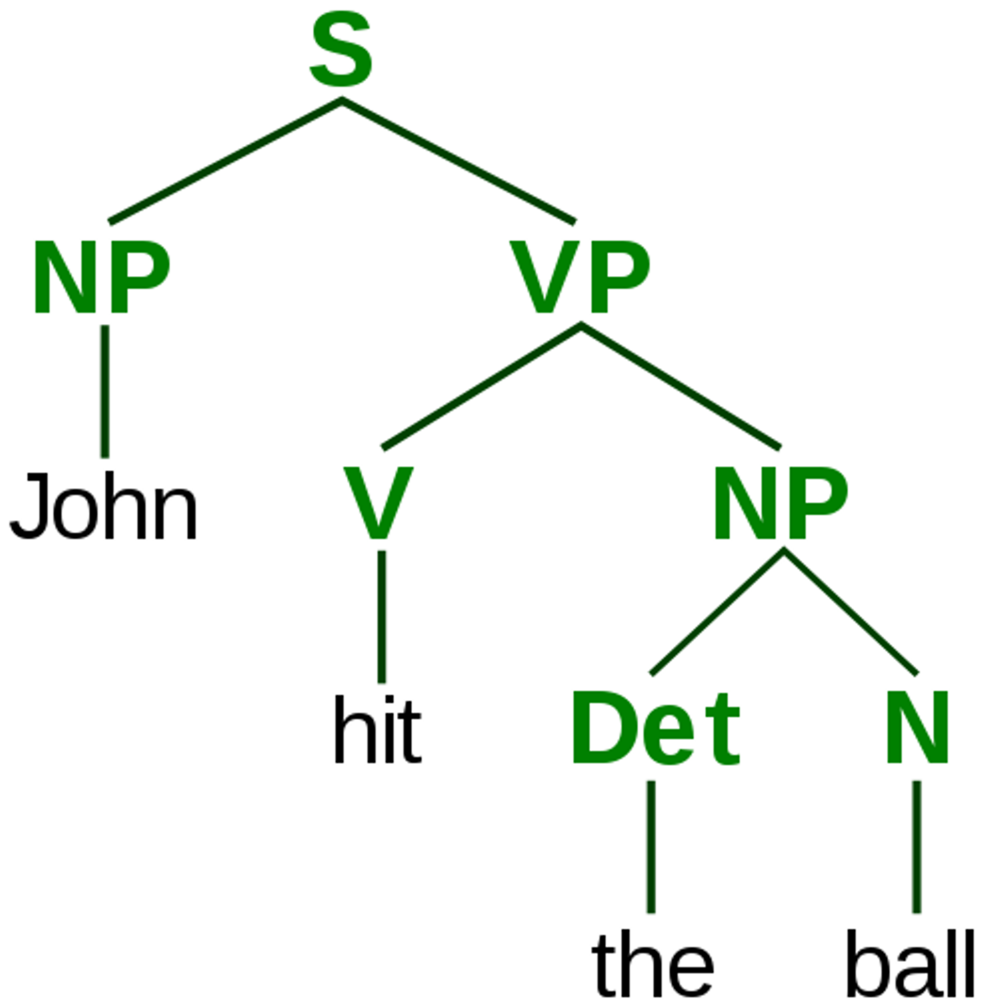
DELIVERABLE

Answers to the above questions

PARSING AND TAGGING

- In order to understand the various elements of a sentence, we need to *tag* important topics and *parse* their dependencies.

- Our goal is to identify the *actors* and *actions* in the text in order to make informed decisions.



PARSING AND TAGGING

- If we are processing financial news, we might need to identify which companies are involved and which actions they are taking.
- If we are writing an assistant application, we might need to identify specific command phrases in order to determine what is being asked:
    - e.g. “Siri, when is my next appointment?”

PARSING AND TAGGING

- Tagging and parsing is made up of a few overlapping subproblems:
  - "Parts of speech" tagging: What are the parts of speech in a sentence (e.g. noun, verb, adjective, etc)?
  - Chunking: Can we identify the pieces of the sentence that go together in meaningful chunks (e.g. noun or verb phrases)?
  - Named entity recognition: Can we identify *specific* proper nouns? Can we pick out people and locations?

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. How might NLP be applied within your current jobs or final projects?
2. What are some other potential NLP use-cases?

DELIVERABLE

Answers to the above questions

TEXT CLASSIFICATION

TEXT CLASSIFICATION

- Text classification is the task of predicting which category or topic a text sample is from.

- For example, we may want to identify whether an article is a sports or business story. Or whether an article has positive or negative sentiment.

- Typically, this is done by using the text as features and the label as the target output. This is referred to as *bag-of-words* classification.

- To include text as features, we usually create a *binary* feature for each word, i.e. does this piece of text contain that word?

TEXT CLASSIFICATION

- To create binary text features, we first create a vocabulary to account for all possible words in our universe.

- As we do this, we need to consider several things.

  - Does order of words matter?

  - Does punctuation matter?

  - Does upper or lower case matter?

TEXT CLASSIFICATION

- This table illustrates features created from the following passage.

"It's a great advantage not to drink among hard drinking people."

Feature	Value
it's	1
great	1
good	0
advantage	1
not	1
think	0
drink	1
from	0
hard	1
drinking	1

Feature	Value
people	1
withhold	0
random	0
smoke	0
among	1
whenever	0
thoughtful	0
inexhaustible	0
men	0
Nick	0

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

Discuss your answers to the following questions and explain your reasoning.

1. Does word order matter?
2. Does word case (e.g. upper or lower) matter?
3. Does punctuation matter?

DELIVERABLE

Answers to the above questions

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. What is “bag-of-words” classification and when should it be used?
2. What are some benefits to this approach?

DELIVERABLE

Answers to the above questions

DEMO

TEXT PROCESSING IN SCIKIT-LEARN

TEXT PROCESSING IN SCIKIT-LEARN

- Scikit-learn has many pre-processing utilities that simplify tasks required to convert text into features for a model.
- These can be found in the `sklearn.preprocessing.text` package.
- We will use the StumbleUpon dataset again to perform text classification. This time, we will use the text content itself to predict whether a page is ‘evergreen’ or not.
- Open the starter code notebook to follow along.

COUNTVECTORIZER

- `CountVectorizer` converts a collection of text into a matrix of features. Each row will be a sample (an article or piece of text) and each column will be a text feature (usually a count or binary feature per word).

- `CountVectorizer` takes a column of text and creates a new dataset. It generates a feature for every word in all of the pieces of text.

- **REMEMBER**: Using all of the words can be useful, but we may need to use *regularization* to avoid overfitting. Otherwise, rare words may cause the model to overfit and not generalize.

COUNTVECTORIZER

- Instantiate a new `CountVectorizer`.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features = 1000,
                             ngram_range=(1, 2),
                             stop_words='english',
                             binary=True)
```

COUNTVECTORIZER PARAMETERS

- There are several parameters to utilize.

- `ngram_range` - a range of word phrases to use
  - `(1,1)` means use all single words
  - `(1,2)` means use all contiguous pairs of word
  - `(1,3)` means use all triples

- `stop_words='english'`
  - Stop words are non-content words (e.g. 'to', 'the', 'it', etc). They aren't helpful for prediction, so they get removed.

COUNTVECTORIZER PARAMETERS

- `max_features=1000`
  - Maximum number of words to consider (uses the first `N` most frequent)

- `binary=True`
  - To use a dummy column as the entry (1 or 0, as opposed to the count). This is useful if you think a word appearing 10 times is no more important than whether the word appears at all.

COUNTVECTORIZER

- Vectorizers are like other models in scikit-learn.
    - We create a vectorizer object with the parameters of our feature space.
    - We `fit` a vectorizer to learn the vocabulary.
    - We `transform` a set of text into that feature space.

COUNTVECTORIZER

- Note: there is a distinction between *`fit`* and *`transform`*.
  - We `fit` from our training set. This is part of the model building process, so we don't look at our test set.
  - We `transform` our test set using our model fit on the training set.

COUNTVECTORIZER EXAMPLE

```
titles = data['title'].fillna('')

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features = 1000,
                             ngram_range=(1, 2),
                             stop_words='english',
                             binary=True)

# Use `fit` to learn the vocabulary of the titles
vectorizer.fit(titles)

# Use `tranform` to generate the sample X word matrix - one column per
feature (word or n-grams)
X = vectorizer.transform(titles)
```

RANDOM FOREST PREDICTION MODEL

- We can now build a random forest model to predict “evergreenness”.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators = 20)

# Use `fit` to learn the vocabulary of the titles vectorizer.fit(titles)

# Use `tranform` to generate the sample X word matrix - one column per feature (word or n-grams)
X = vectorizer.transform(titles)
y = data['label']

from sklearn.cross_validation import cross_val_score

scores = cross_val_score(model, X, y, scoring='roc_auc')
print('CV AUC {}, Average AUC {}'.format(scores, scores.mean()))
```

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- An alternative *bag-of-words* approach to `CountVectorizer` is a Term Frequency - Inverse Document Frequency (TF-IDF) representation.

- TF-IDF uses the product of two intermediate values, the *Term Frequency* and *Inverse Document Frequency*.

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- *Term Frequency* is equivalent to `CountVectorizer` features, just the number of times a word appears in the document (i.e. count).

- *Document Frequency* is the percentage of documents that a particular word appears in.

- For example, "the" would be 100% while "Syria" is much lower.

- *Inverse Document Frequency* is just 1/Document Frequency.

TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

- Combining, TF-IDF = Term Frequency * Inverse Document Frequency or TF-IDF = Term Frequency / Document Frequency

- The intuition is that the words that have high weight are those that either appear ***frequently*** in this document or appear ***rarely*** in other documents (and are therefore unique to this document).

- This is a good alternative to using a static set of “stop” words.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
```

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. What does TF-IDF stand for?
2. What does this function do and why is it useful?
3. Use `TfidfVectorizer` to create a feature representation of the StumbleUpon titles.

DELIVERABLE

Answers to the above questions and feature representation

CONCLUSION

TOPIC REVIEW

LET'S REVIEW

- Natural language processing (NLP) is the task of pulling meaning and information from text.

- This typically involves many subproblems including tokenization, cleaning (stemming and lemmatization), and parsing.

- After we have structured our text, we can identify features for other tasks, including classification, summarization, and translation.

- In scikit-learn, we use vectorizers to create text features for classification, such as `CountVectorizer` and `TfIdfVectorizer`.