

# NLP AND TEXT CLASSIFICATION

*Jonathan Balaban*

*DAT<sub>2</sub>*

---

## NLP 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

---

# PRE-WORK REVIEW

---

- ▶ Decision Trees and Random Forests
- ▶ Experience with scikit-learn classifiers, specifically random forests and decision trees
- ▶ Install the Python package spacy with `pip install spacy`
- ▶ Run the spacy download data command

```
python -m spacy.en.download --force all
```

---

# WHAT IS NATURAL LANGUAGE PROCESSING (NLP)?

---

- ▶ The task of extracting information and meaning from text documents.
  - ▶ There are many types of information.
- ▶ 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:
  - ▶ Unusual punctuation (common in social media)
  - ▶ Different language conventions.
  - ▶ E.g. The L.A. Lakers won the NBA championship in 2010, defeating the Boston Celtics.

---

# TOKENIZATION EXAMPLES

---

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

The Falcons are my team. → [The, Falcons, are, my, 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

shouted → shout

best → good

better → good

good → good

wiping → wipe

hidden → hide

## Stemming

badly → bad

computing → comput

computed → comput

wipes → wip

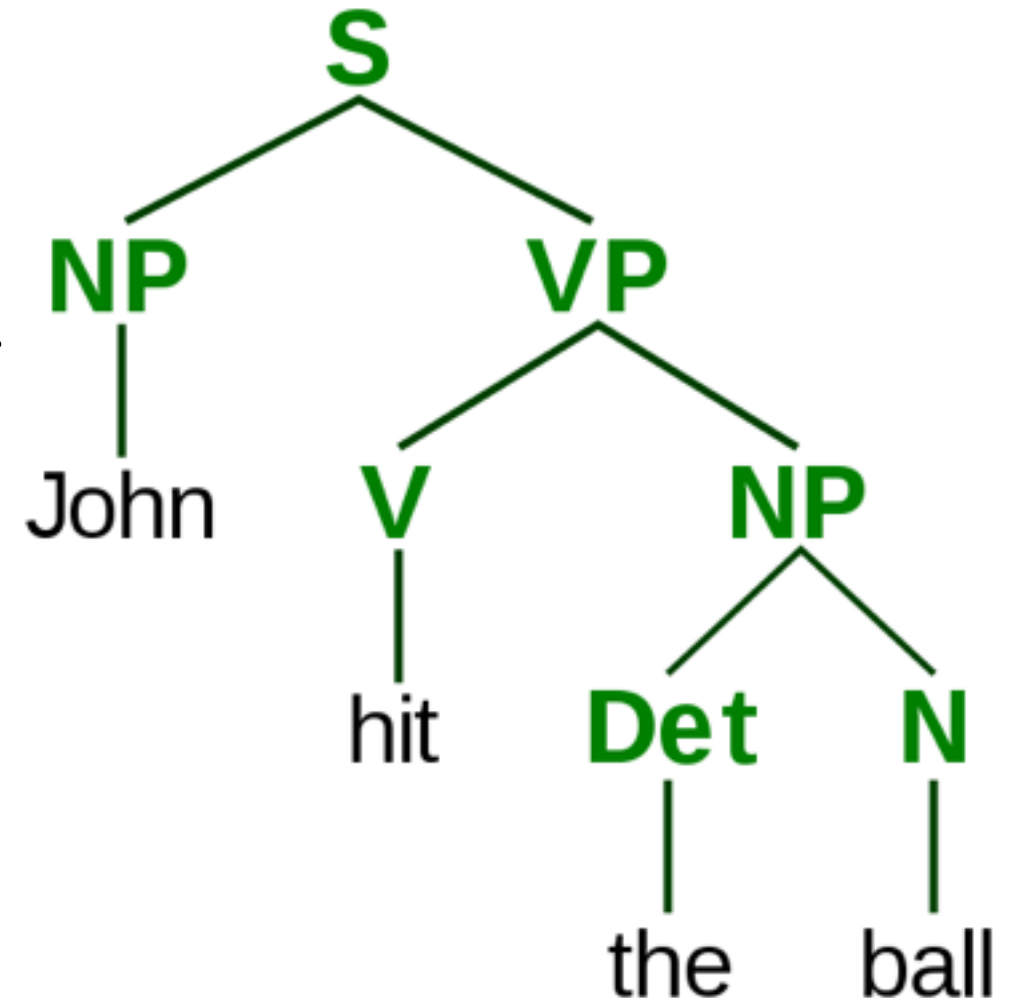
wiped → wip

wiping → wip

# 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 any 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?

---

**DEMO**

---

# NLP WITH 'SPACY'

---

# NATURAL LANGUAGE PROCESSING WITH ‘SPACY’

---

- ▶ Most NLP techniques require pre-processing large collections of annotated text in order to learn specific language rules.
- ▶ There are many tools available for English and other popular languages.
- ▶ Each tool typically requires a large amount of data and large databases of special use-cases like language inconsistencies and slang.
- ▶ In Python, two popular NLP packages are `nltk` and `spacy`.
- ▶ `nltk` is more popular but not as advanced and well maintained. `spacy` is more modern but not available for commercial use.

---

# NATURAL LANGUAGE PROCESSING WITH 'SPACY'

---

- ▶ We'll be using spacy to process some news article titles. First load the NLP toolkit by specifying the language.

```
from spacy.en import English  
nlp_toolkit = English()
```

- ▶ This toolkit has 3 pre-processing engines:
  - ▶ A tokenizer: to identify the word tokens
  - ▶ A tagger: to identify the concepts described by the words
  - ▶ A parser: to identify the phrases and links between different words
- ▶ Each of these engines can be overridden with a different, specialized tool. You can even write your own and use them in place of the defaults.



---

# NATURAL LANGUAGE PROCESSING WITH ‘SPACY’

---

- ▶ The first title is “[IBM Sees Holographic Calls, Air Breathing Batteries](#)”.
- ▶ Let’s extract several pieces of information: this title references a company that is postulating a new possible product, air-breathing batteries.



---

# NATURAL LANGUAGE PROCESSING WITH 'SPACY'

---

- We can use spacy to get information about this title.

```
title = "IBM sees holographic calls, air breathing batteries"
parsed = nlp_toolkit(title)
```

```
for (i, word) in enumerate(parsed):
    print("Word: {}".format(word))
    print("\t Phrase type: {}".format(word.dep_))
    print("\t Is the word a known entity type? {}".format(word.ent_type_) if
word.ent_type_ else "No"))
    print("\t Lemma: {}".format(word.lemma_))
    print("\t Parent of this word: {}".format(word.head.lemma_))
```

- `nlp_toolkit` runs each of the individual pre-processing tools.

---

# NATURAL LANGUAGE PROCESSING WITH ‘SPACY’

---

► The output will look similar to this:

Word: IBM

Phrase type: nsubj

Is the word a known entity type? ORG

Lemma: ibm

Parent of this word: see

Word: sees

Phrase type: ROOT

Is the word a known entity type? No

Lemma: see

Parent of this word: see

Word: holographic

Phrase type: amod

Is the word a known entity type? No

Lemma: holographic

Parent of this word: call

---

# NATURAL LANGUAGE PROCESSING WITH ‘SPACY’

---

- ▶ In this output:
  - ▶ “IBM” is identified as an organization (ORG).
  - ▶ We identify a phrase “holographic calls”.
  - ▶ We identify a compound noun phrase, “air breathing batteries”.
  - ▶ We identify that “see” is at a root as it’s the action “IBM” is taking.
  - ▶ We can see that “batteries” was lemmatized to “battery”.

---

# NATURAL LANGUAGE PROCESSING WITH 'SPACY'

---

- We can use this output to find all titles that discuss an organization.

```
def references_organization(title):  
    parsed = nlp(title)  
    return any([word.ent_type_ == 'ORG' for word in parsed])  
  
data['references_organization'] =  
data['title'].fillna('').map(references_organization)  
  
data[data['references_organization']][['title']].head()
```

---

# COMMON PROBLEMS IN NLP

---

- ▶ These subtasks are very difficult, because language is complex and ever changing.
- ▶ Most often, we are looking for heuristics to search through large amounts of text data. The results may not be perfect and that's okay.
- ▶ Older techniques rely on rule-based systems; more recent research has focused on more flexible systems, focusing on words used rather than sentence structure.
- ▶ We'll see an example of these modern approaches in the next class.

---

## INTRODUCTION

---

# TEXT CLASSIFICATION

---

# TEXT CLASSIFICATION

---

- ▶ Text classification predicts what category or topic a piece of text is from.
  - ▶ We may want to identify whether an article is a sports or business story, or positive vs. 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

---

► 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

---

## ANSWER THE FOLLOWING QUESTIONS



EXERCISE

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?

---

# 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.

---

# 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

---

- ▶ `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 `transform` 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 `transform` 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  $1/\text{Document Frequency}$ .

---

# TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY

---

- ▶ Combining,  $TF\text{-}IDF = \text{Term Frequency} * \text{Inverse Document Frequency}$  or  $TF\text{-}IDF = \text{Term Frequency} / \text{Document Frequency}$
- ▶ 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

---

## ANSWER THE FOLLOWING QUESTIONS



EXERCISE

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.

---

# ACTIVITY: TEXT CLASSIFICATION IN SCIKIT-LEARN

---



## EXERCISE

### DIRECTIONS (30 minutes)

1. Use the text features of `title` with one or more feature sets from the previous random forest model. Train this model to see if it improves AUC.
2. Use the body text instead of the `title`. Does this give an improvement?
3. Use `TfidfVectorizer` instead of `CountVectorizer`. Does this give an improvement?

**Check:** Were you able to prepare a model that uses both quantitative features and text features? Does this model improve the AUC?



---

# 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`.

---

**COURSE**

---

**BEFORE NEXT  
CLASS**

**Final: Part 2**

# PROJECT DUE DATES

Unit	Project	Assigned	Deadline
Unit 1	Project 1	Assigned Lesson 1	Due Lesson 3
Unit 1	Project 2	Assigned Lesson 3	Due Lesson 5
Unit 2	Final Project, pt 1	Assigned Lesson 1	Due Lesson 8
Unit 2	Project 3	Assigned Lesson 5	Due Lesson 9
Unit 2	Project 4	Assigned Lesson 9	Due Lesson 11
Unit 3	Final Project, pt 2	Assigned Lesson 8	Due Lesson 14
Unit 3	Final Project, pt 3	Assigned Lesson 14	Due Lesson 16
Unit 3	Final Project, pt 4	Assigned Lesson 16	Due Lesson 18
Unit 3	Final Project, pt 5	Assigned Lesson 18	Due Lesson 20

---

# LESSON

---

# Q & A

## LESSON

---

# EXIT TICKET

**DON'T FORGET TO FILL OUT YOUR EXIT  
TICKET**