

# Learning Objectives

After this lesson, you should be able to:

- Define natural language processing
- List common tasks associated with
  - Use-cases
  - Tokenization
  - Stemming and lemmatization
  - Tagging and parsing
- Demonstrate how to classify text or documents using scikit-learn

# End-of-Course/Projects Countdown

Unit Project 4	June 21; due next session
Final Project, Part 4	June 30; due in 2 weeks
Final Project, Part 5	July 8; due in 3 weeks

The logo consists of a solid black circle containing the white letters "DS" in a bold, sans-serif font.

**DS**

Q & A



DS

# Review

A black circle containing the white text "DS".

DS

# Review

*Decision Trees and Random Forests*

A black circle containing the white text "DS".

DS

# Today

# Here's what's happening today:

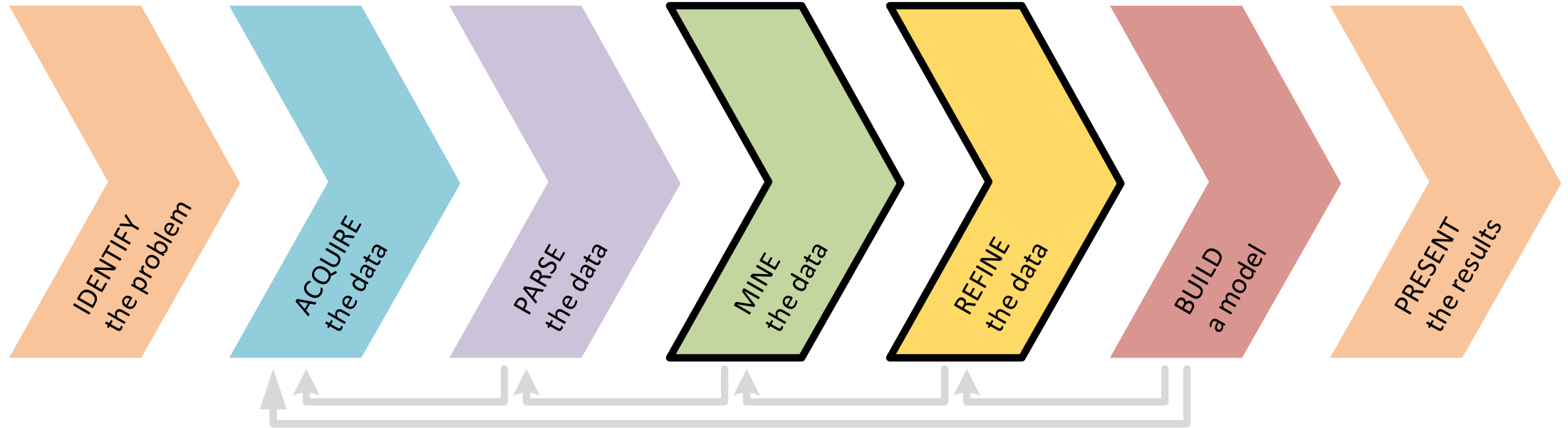
- Announcements and Exit Tickets
- Review
- Natural Language Processing
  - Understanding and generation
  - NLP is hard...
  - Tokenization
  - Stemming and lemmatization
  - Tagging and parsing
  - Demo – Tokenizing, tagging, and parsing with *spacy*
- Text Classification
  - Bag-of-words classification
  - Text Processing with *sklearn*
  - Term-Frequency and Inverse-Document-Frequency (TF-IDF)
- Lab – Natural Language Processing and Text Classification
- Office hours during class for final projects
- Review

# Where does Natural Language Processing fits in the course?

<b>Research Design and Data Analysis</b>	Research Design	Data Visualization in <i>pandas</i>	Statistics	Exploratory Data Analysis in <i>pandas</i>
<b>Foundations of Modeling</b>	Linear Regression	Classification Models	Evaluating Model Fit	Presenting Insights from Data Models
<b>Data Science in the Real World</b>	Decision Trees and Random Forests	Time Series Data	Natural Language Processing	Databases



Therefore, today we are refocusing on the **MINE** the **data** and **REFINE** the **data** steps but with a focus on natural language processing and text classification



The logo consists of a solid black circle containing the white letters "DS" in a bold, sans-serif font.

**DS**

Q & A



**DS**

# Pre-Work

# Pre-Work

Before this lesson, you should already be able to:

- Experience with *sklearn* classifiers, specifically random forests and decision trees
- Install the Python package *spacy* with `conda install spacy`
- Run the *spacy* download data command with `python -m spacy.en.download --force all`

DS

# Natural Language Processing

# What is Natural Language Processing?

- Natural Language Processing (NLP) is the study of the computational treatment of natural (human) language, i.e., teaching computers how to understand (and generate) human language

# Basic NLP Pipeline: Understanding and Generation



# NLP is tasked with extracting meaning and information from (text) documents

## Understanding

- 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

## Generation

- 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



# What are some real-world examples of NLP?

- Search engines (E.g., Google and Bing)
- Natural language assistants (E.g., Apple's Siri uses voice recognition to record a command and then various fairly advanced NLP engines to identify the question asked and possible answers)
- Machine translation (E.g., Google Translate)
- Question answering (E.g., IBM's Watson)
- News digest (E.g., Yahoo!)

# Computers are confused by (human) language

- E.g., “Children make delicious snacks”

‣ *Are* Children delicious snacks?

‣ Do children *prepare* delicious snacks?

Each genre of text (e.g., blogs, emails, press releases, chats) presents different challenges to NLP

- E.g., newspapers news headlines
  - “Red tape holds up new bridges”
  - “Government head seeks arms”
  - “Blair wins on budget, more lies ahead”

DS

# Natural Language Processing

*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
- What sort of difficulties may there be with the following sentence?
  - “The L.A. Lakers won the NBA championship in 2010, defeating the Boston Celtics”

“The L.A. Lakers won the NBA championship in 2010, defeating the Boston Celtics”

- ▶ To perform a proper analysis, we need to be able to identify that:
  - ▶ The periods in “L.A.” don’t mark the end of a sentence but an abbreviation
  - ▶ “L.A. Lakers” and “Boston Celtics” are one concept.
  - ▶ “2010” is the word used, not “2010,”

# Tokenization Examples

Sentence	Tokens
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]

DS

# Natural Language Processing

*Stemming and Lemmatization*



Stemming and lemmatization help identify common roots of words

- How would you describe the relationship between the terms ‘bad’ and ‘badly’ or ‘different’ and ‘differences’?

# Stemming is a crude process of removing common endings from words

- To stem a word is to reduce it to a base form, called the *stem*, after removing various suffixes and endings and, sometimes, performing some additional transformations
- In practice, prefixes are sometimes preserved, so ‘rescan’ will not be stemmed to ‘scan’

- E.g.,
  - badly → bad
  - computing → comput
  - computed → comput
  - wipes → wip
  - wiped → wip
  - wiping → wip

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 'best' and 'better'

- E.g.,
  - best → good
  - better → good
  - good → good
  - wiping → wipe
  - hidden → hide
  - shouted → shout

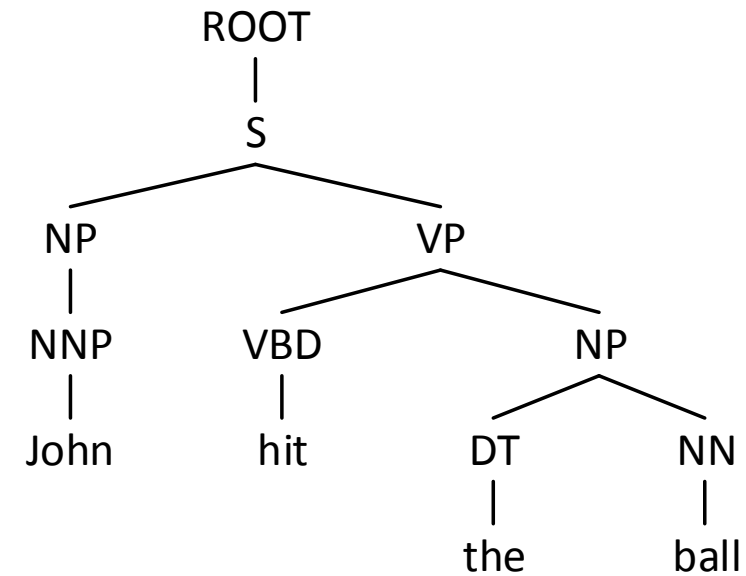
DS

# Natural Language Processing

*Tagging and Parsing*

# Tagging and Parsing

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



DT - Determiner  
NN - Noun, singular or mass  
NNP - Proper noun, singular  
NP - Noun phrase  
S - Simple declarative clause  
VBD - Verb, past tense  
VP - Verb phrase

# Tagging and Parsing (cont.)

- Our goal is to identify the *actors* and *actions* in the text in order to make informed decisions
- E.g., if we are processing financial news, we might need to identify which companies are involved and any actions they are taking
- E.g., 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?”)

# Tagging and parsing is made up of a few overlapping subtasks

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

## Tagging

John/NNP hit/VBD the/DT ball/NN

## Parsing

```
(ROOT
  (S
    (NP (NNP John))
    (VP (VBD hit)
      (NP (DT the) (NN ball))))))
```

# 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
- These techniques rely on rule-based systems but more recent research has focused on more flexible systems, focusing on words used rather than on the structure of the sentence
- We'll see an example of these modern approaches in the next class



A black circle containing the white text "DS".

DS

# Natural Language Processing

*Demo – Tokenizing, Tagging, and Parsing with spacy*

# Natural Language Processing in Python

- 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

# Tokenizing, tagging, and parsing with *spacy*

- *spacy* 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

# We'll be using *spacy* to tokenize, tag, and parse “John hit the ball”

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

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

```
sentence = u'John hit the ball'
parsed = nlp_toolkit(sentence)
```

```
for (i, word) in enumerate(parsed):
    print word
    print "\tParent: {}".format(word.head.lemma_)
    print "\tPhrase type: {}".format(word.dep_)
    print "\tKnown entity type: {}".format(word.ent_type_ if word.ent_type_ else 'n/a')
    print "\tLemma: {}".format(word.lemma_)
```

# “John hit the ball” after tokenization, tagging, and parsing

John

Parent: hit  
Phrase type: nsubj  
Known entity type: PERSON  
Lemma: john

hit

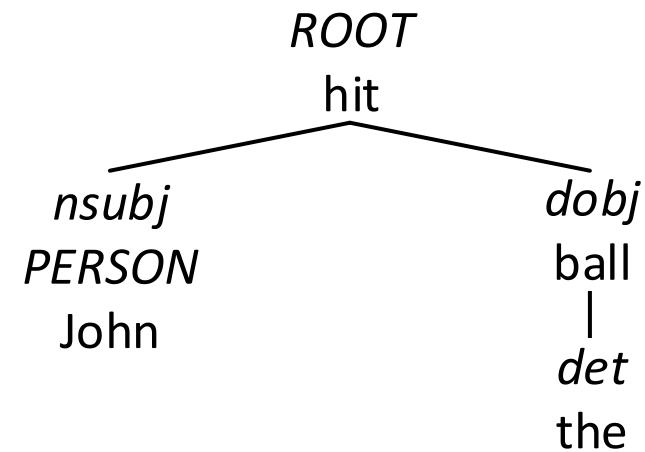
Parent: hit  
Phrase type: ROOT  
Known entity type: n/a  
Lemma: hit

the

Parent: ball  
Phrase type: det  
Known entity type: n/a  
Lemma: the

ball

Parent: hit  
Phrase type: dobj  
Known entity type: n/a  
Lemma: ball

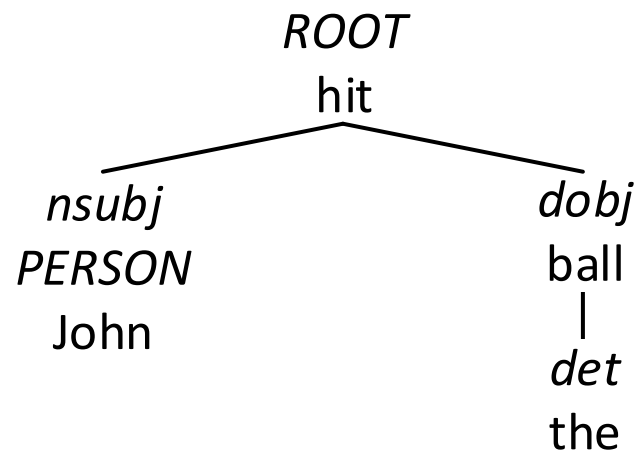


nsubj: link between a verb and an NP subject  
e.g. 'Clinton defeated Dole' - nsubj(defeated,Clinton)

dobj: link between a verb and one of its accusative objects  
e.g. 'she gave me a raise' - dobj(gave,raise)

det: link from a noun to its determiner  
e.g. 'the man' - det(man,the), 'which man' - det(man,which)

# “John hit the ball” after tokenization, tagging, and parsing (cont.)



nsubj: link between a verb and an NP subject  
e.g. 'Clinton defeated Dole' - nsubj(defeated,Clinton)

dobj: link between a verb and one of its accusative objects  
e.g. 'she gave me a raise' - dobj(gave,raise)

det: link from a noun to its determiner  
e.g. 'the man' - det(man,the), 'which man' - det(man,which)

- ▶ In this output,
  - ▶ “John” is identified as a person (PERSON)
  - ▶ We identify that “hit” is at its root as it is the action “John” took

A black circle containing the white text "DS".

DS

# Text Classification

# Text Classification

- Text classification is the task of predicting what category or topic a piece of text is from
- For example, we may want to identify whether an article is a sports or business story
- Or whether has positive or negative sentiment



# Text Classification

- 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?
- As we do this, we need to consider several things
  - Does order of words matter?
  - Does punctuation matter?
  - Does upper or lower case matter?

# “John hit the ball”

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

$$x = (x_{aardvark}, \dots, x_{ball}, \dots, x_{hit}, \dots, x_{John}, \dots, x_{the}, \dots, x_{zyzzogeton})$$

- ▶ “John hit the ball”

$$x = \left( \underbrace{\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}}_{0}, \underbrace{x_{ball}}_1, \underbrace{\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}}_{0}, \underbrace{x_{hit}}_1, \underbrace{\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}}_{0}, \underbrace{x_{John}}_1, \underbrace{\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}}_{0}, \underbrace{x_{the}}_1, \underbrace{\begin{smallmatrix} \vdots \\ 0 \end{smallmatrix}}_{0} \right)$$

A black circle containing the white text 'DS' in a bold, sans-serif font.

DS

# Text Classification

*Codealong - Text Processing with sklearn*

# Text Classification

*Term Frequency and Inverse Document Frequency (TF-IDF)*

# TF-IDF

- 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 (TF)

- *Term Frequency* is equivalent to `CountVectorizer` features but using frequencies, not counts

$$tf(t, d) = \frac{\text{number of occurrences of term } t \text{ in document } d}{\text{number of terms in document } d}$$

- *Term Frequency* assigns high weight to frequent words (words that appear frequently) in a documents

# Inverse Document Frequency (IDF)

- *Document Frequency* is the percentage of documents that a particular word appears in
- *Inverse Document Frequency* is *Document Frequency*'s inverse

$$idf(t, D) = \frac{\text{total number of documents } D}{\text{number of documents term } t \text{ appears in them}}$$

- *Inverse Document Frequency* assigns high weight to rare words in all the documents

## TF-IDF (cont.)

- Combining,

$$tf-idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

- The intuition behind *TF-IDF* is to assign high weight to words that either
  - appear frequently in this document or
  - appear rarely in other documents (and are therefore specific to this document)





# Lab

*Natural Language Processing and Text Classification*

A black circle containing the white text "DS".

DS

# Review

# Review

- Natural language processing (NLP) is the task of pulling meaning and information from text
- This typically involves many sub problems 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`

**DS**

Q & A



**DS**

# Before Next Class

# Before Next Class

Before the next lesson, you should already be able to:

- › Install *gensim* with `conda install gensim`
- › Recall and apply *unsupervised learning* techniques
- › Recall probability distributions, specifically discrete multinomial distributions
- › Recall NLP essentials, including experience with *spacy*
- › BONUS: Setup Twitter API credentials using the provided instructions

# Next Class

*Latent Variables and Natural Language Processing*

# Learning Objectives

After the next lesson, you should be able to:

- Understand what *latent variables* are
- Understand the uses of *latent variables* in language processing
- Use the *word2vec* and LDA algorithms of *gensim*





DS

# Exit Ticket

*Don't forget to fill out your exit ticket [here](#)*

# Sources

- Introduction to Natural Language Processing, University of Michigan