Text Classification Part 2: Preprocessing and Feature Engineering

CSCI 1460: Computational Linguistics

Lecture 3

Ellie Pavlick

Fall 2023

Topics

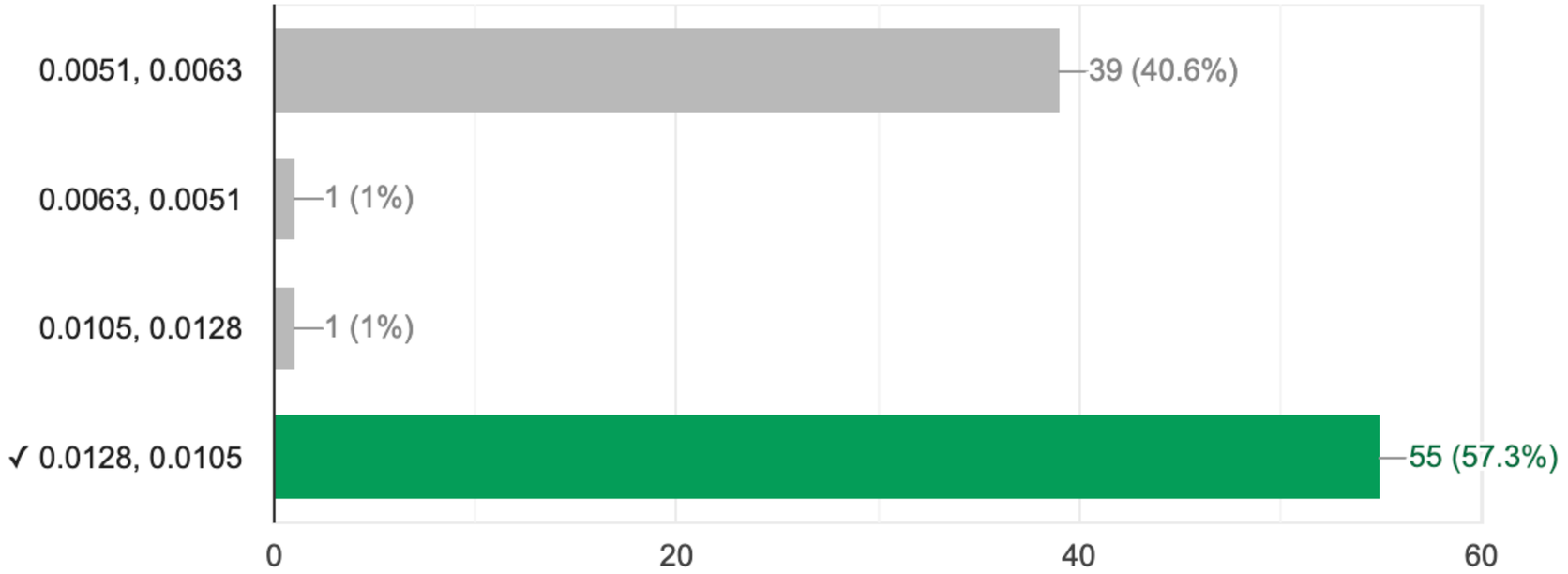
- Train-Test Splits and Baselines
- Preprocessing
- Feature Engineering
  - Weighting Strategies
  - Ngrams
  - More Advanced Features

Quiz Recap

Below are the parameters for a Naive Bayes sentiment classifier. What is the evidence (P(X|Y)) for the positive and negative classes, respectively?

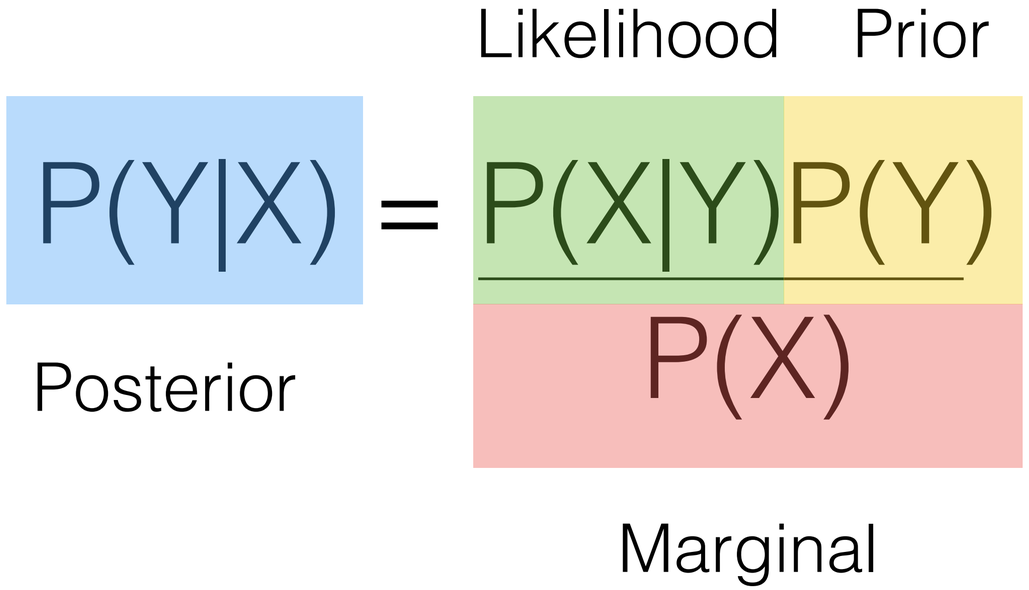
Copy

55 / 96 correct responses



Naive Bayes Classifiers

Bayes Rule



Quiz Recap

- Lots of questions about the x vector in logistic regression!
- (My slide used made up numbers, sorry!)

Logistic Regression Classifiers

Inference

Logistic Regression

	w
Tax	0.5
This	0.5
600	1.0
👇	1.0
Awesome	0.6
Policies	0.2

600 Awesome Tax Policies

???

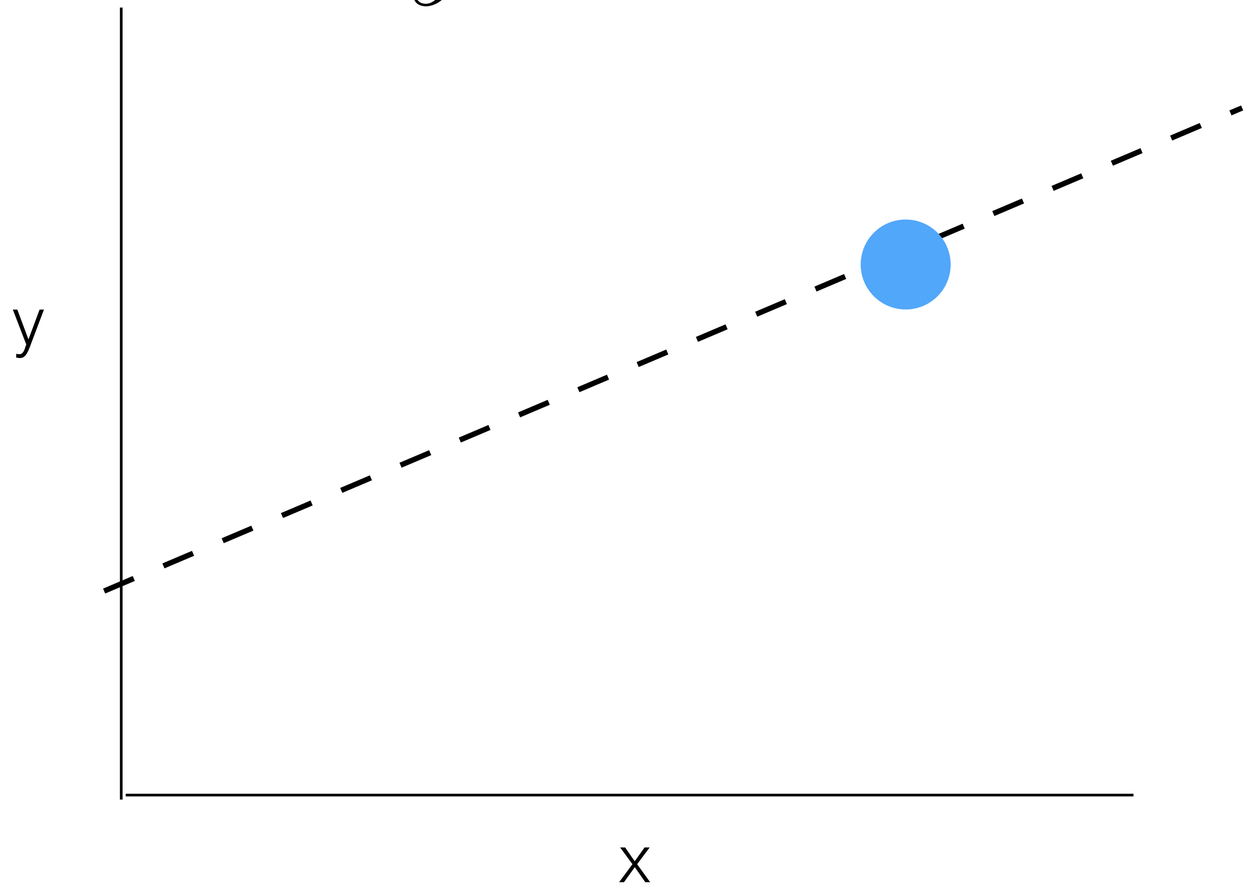
Logistic Regression Classifiers

Inference

	w
Tax	0.5
This	0.5
600	1.0
👇	1.0
Awesome	0.6
Policies	0.2

600 Awesome Tax Policies

$$y = \vec{w} \cdot \vec{x}$$



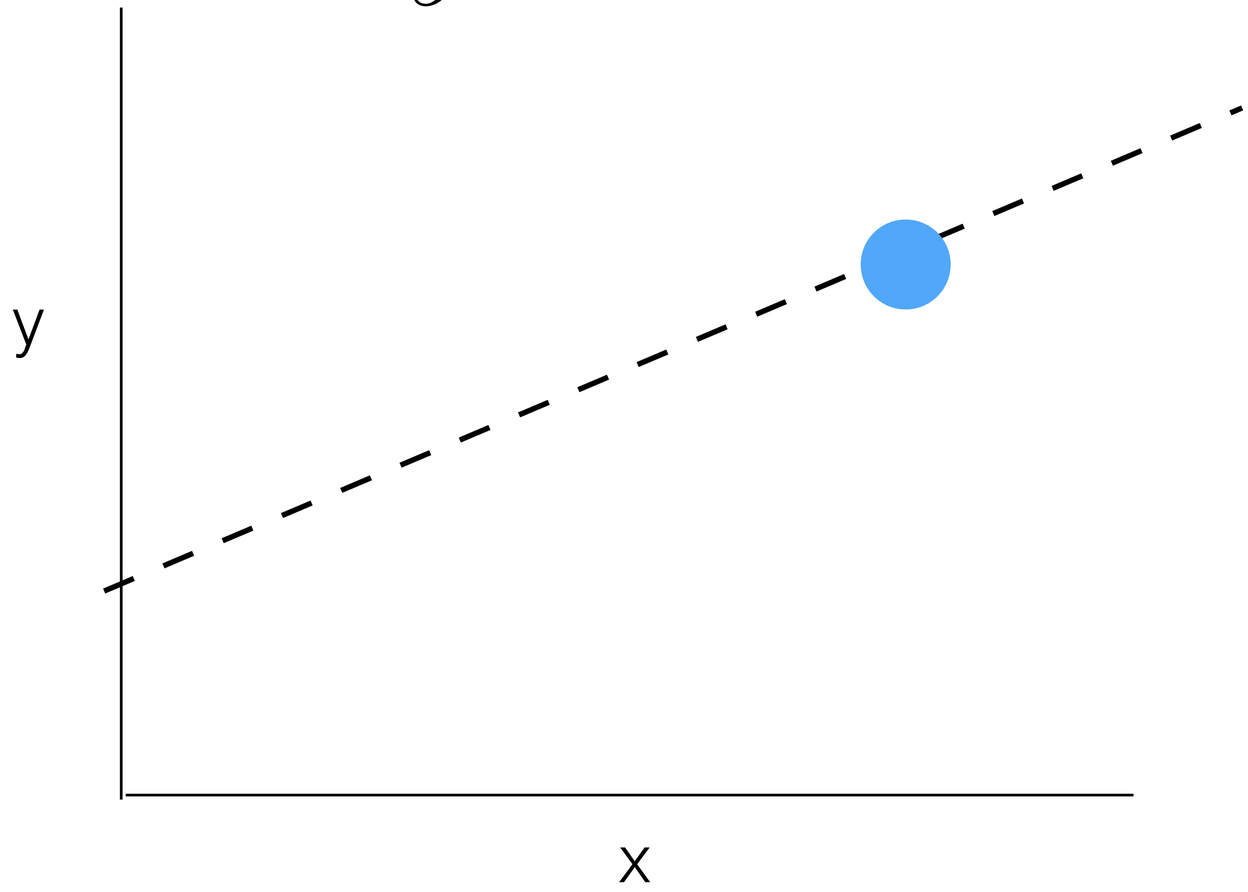
Logistic Regression Classifiers

Inference

	w	x
Tax	0.5	1
This	0.5	0
600	1.0	1
👇	1.0	0
Awesome	0.6	1
Policies	0.2	1

600 Awesome Tax Policies

$$y = \vec{w} \cdot \vec{x}$$



Logistic Regression Classifiers

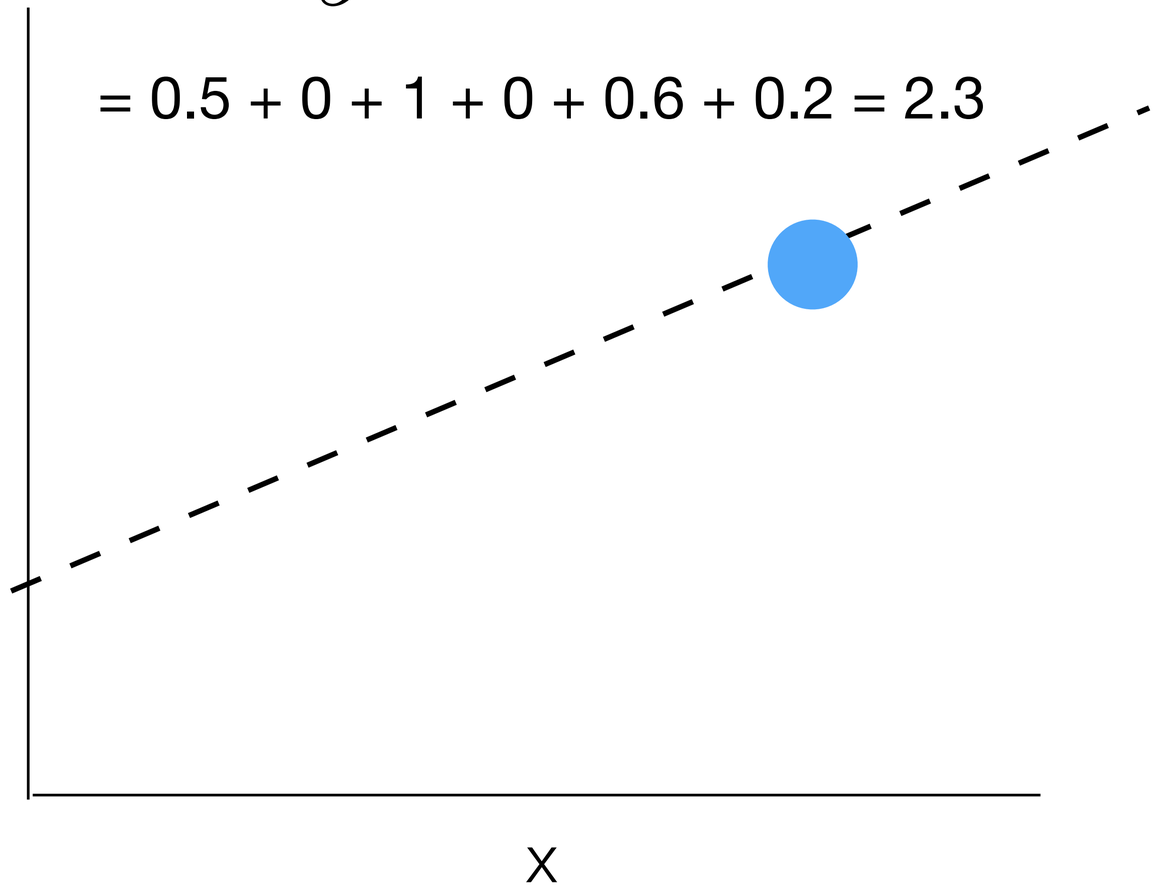
Inference

	w	x
Tax	0.5	1
This	0.5	0
600	1.0	1
👇	1.0	0
Awesome	0.6	1
Policies	0.2	1

600 Awesome Tax Policies

$$y = \vec{w} \cdot \vec{x}$$

$$= 0.5 + 0 + 1 + 0 + 0.6 + 0.2 = 2.3$$



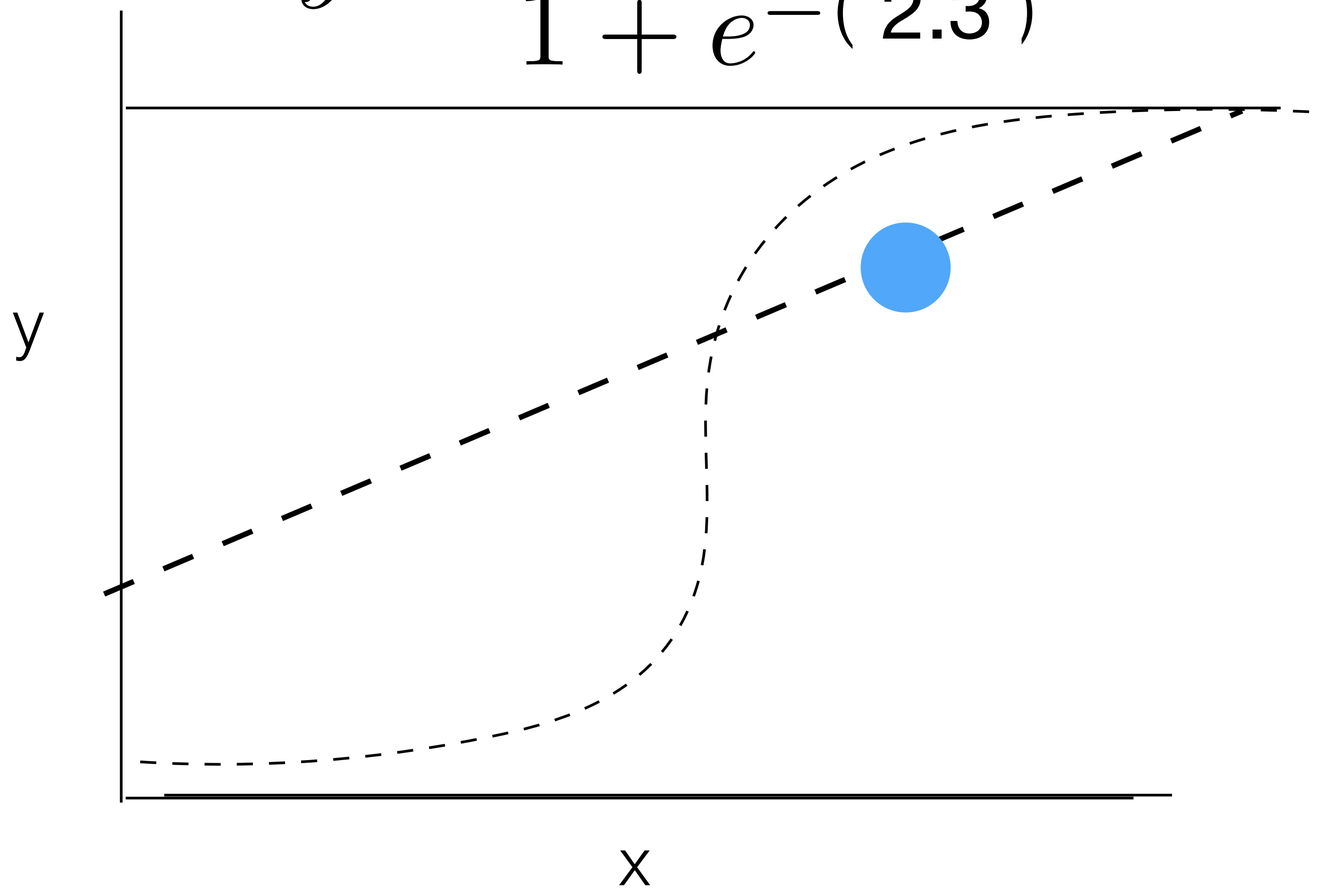
Logistic Regression Classifiers

Inference

x	w
Tax	0.5
This	0.5
600	1.0
👇	1.0
Awesome	0.6
Policies	0.2

600 Awesome Tax Policies

$$y = \frac{1}{1 + e^{-(2.3)}}$$



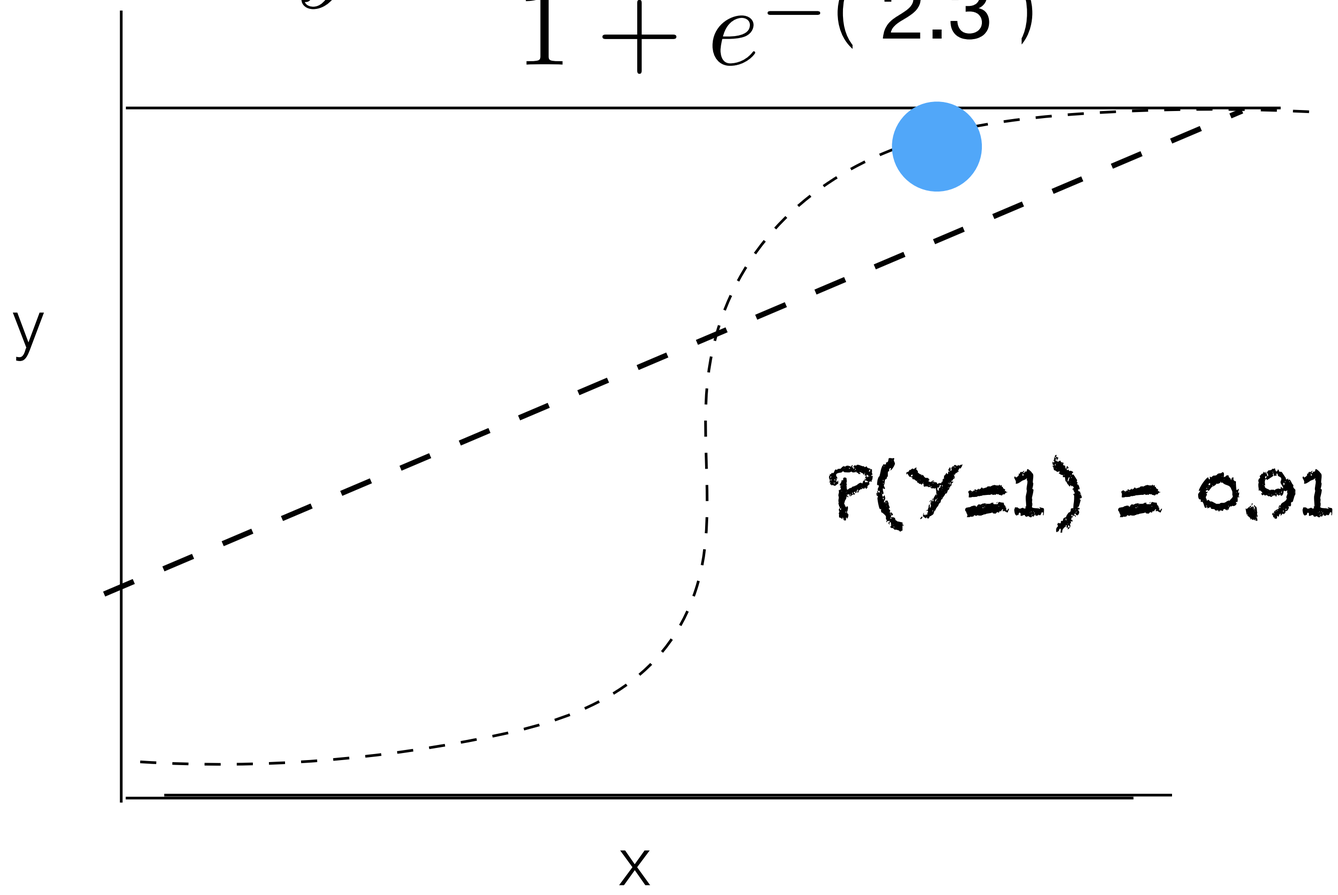
Logistic Regression Classifiers

Inference

x	w
Tax	0.5
This	0.5
600	1.0
👇	1.0
Awesome	0.6
Policies	0.2

600 Awesome Tax Policies

$$y = \frac{1}{1 + e^{-(2.3)}}$$



Topics

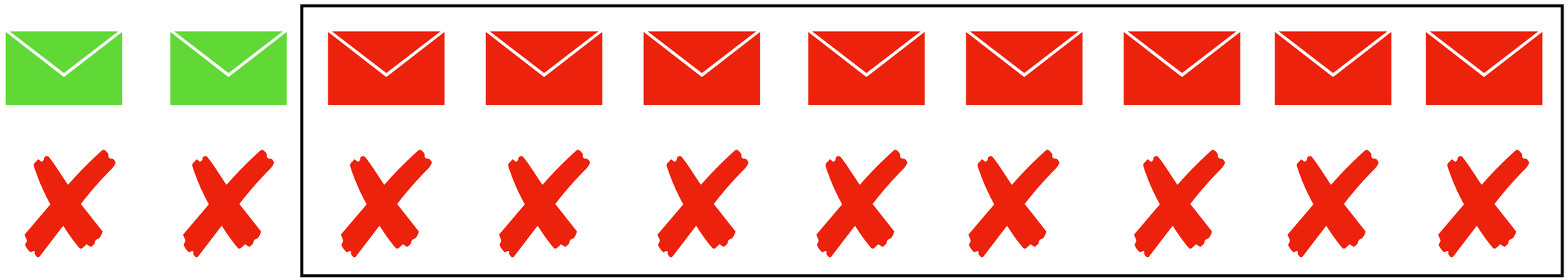
- **Train-Test Splits and Baselines**
- Preprocessing
- Feature Engineering
  - Weighting Strategies
  - Ngrams
  - More Advanced Features

Baselines

- Baseline: A simpler model/way of solving the problem
  - Used to put results in context
- E.g., 80% sounds pretty good, but if “always predict spam” gets 80% accuracy, then an ML model which gets 80% is not very impressive…

Baselines

- Baseline: A simpler model/way of solving the problem
  - Used to put results in context
- E.g., 80% sounds pretty good, but if “always predict spam” gets 80% accuracy, then an ML model which gets 80% is not very impressive…



Baselines

- Common baselines to report:
  - Random: Guess at random
  - “Most frequent class”: For classification tasks, always predict whichever label is most common in the training set
  - Prior state-of-the-art (SOTA): i.e., the “defending champ”
  - Various task-specific heuristics, e.g.,
    - For QA: pick the first name in the passage
    - For IR: sort documents according to length
    - Usually requires some creativity

Baselines

SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pranav Rajpurkar and **Jian Zhang** and **Konstantin Lopyrev** and **Percy Liang**
{pranavsr,zjian,klopyrev,pliang}@cs.stanford.edu
Computer Science Department
Stanford University

	Exact Match		F1	
	Dev	Test	Dev	Test
Random Guess	1.1%	1.3%	4.1%	4.3%
Sliding Window	13.2%	12.5%	20.2%	19.7%
Sliding Win. + Dist.	13.3%	13.0%	20.2%	20.0%
Logistic Regression	40.0%	40.4%	51.0%	51.0%
Human	80.3%	77.0%	90.5%	86.8%

Table 5: Performance of various methods and humans. Logistic regression outperforms the baselines, while there is still a significant gap between humans.

Baselines

SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pranav Rajpurkar and **Jian Zhang** and **Konstantin Lopyrev** and **Percy Liang**
{pranavsr,zjian,klopyrev,pliang}@cs.stanford.edu
Computer Science Department
Stanford University

	Exact Match		F1	
	Dev	Test	Dev	Test
Random Guess	1.1%	1.3%	4.1%	4.3%
Sliding Window	13.2%	12.5%	20.2%	19.7%
Sliding Win. + Dist.	13.3%	13.0%	20.2%	20.0%
Logistic Regression	40.0%	40.4%	51.0%	51.0%
Human	80.3%	77.0%	90.5%	86.8%

Table 5: Performance of various methods and humans. Logistic regression outperforms the baselines, while there is still a significant gap between humans.

Baselines

SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pranav Rajpurkar and **Jian Zhang** and **Konstantin Lopyrev** and **Percy Liang**
{pranavsr,zjian,klopyrev,pliang}@cs.stanford.edu
Computer Science Department
Stanford University

	Exact Match		F1	
	Dev	Test	Dev	Test
Random Guess	1.1%	1.3%	4.1%	4.3%
Sliding Window	13.2%	12.5%	20.2%	19.7%
Sliding Win. + Dist.	13.3%	13.0%	20.2%	20.0%
Logistic Regression	40.0%	40.4%	51.0%	51.0%
Human	80.3%	77.0%	90.5%	86.8%

Table 5: Performance of various methods and humans. Logistic regression outperforms the baselines, while there is still a significant gap between humans.

Baselines

SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pranav Rajpurkar and **Jian Zhang** and **Konstantin Lopyrev** and **Percy Liang**
{pranavsr,zjian,klopyrev,pliang}@cs.stanford.edu
Computer Science Department
Stanford University

	Exact Match		F1	
	Dev	Test	Dev	Test
Random Guess	1.1%	1.3%	4.1%	4.3%
Sliding Window	13.2%	12.5%	20.2%	19.7%
Sliding Win. + Dist.	13.3%	13.0%	20.2%	20.0%
Logistic Regression	40.0%	40.4%	51.0%	51.0%
Human	80.3%	77.0%	90.5%	86.8%

Table 5: Performance of various methods and humans. Logistic regression outperforms the baselines, while there is still a significant gap between humans.

Baselines

- “Skylines”: upper bounds on performance
- Common skylines:
  - Human performance on the task
  - Performance under ideal conditions (e.g., how good would my QA system be if we tell it which sentence to look at…)

Baselines

SQuAD: 100,000+ Questions for Machine Comprehension of Text

Pranav Rajpurkar and **Jian Zhang** and **Konstantin Lopyrev** and **Percy Liang**
{pranavsr,zjian,klopyrev,pliang}@cs.stanford.edu
Computer Science Department
Stanford University

	Exact Match		F1	
	Dev	Test	Dev	Test
Random Guess	1.1%	1.3%	4.1%	4.3%
Sliding Window	13.2%	12.5%	20.2%	19.7%
Sliding Win. + Dist.	13.3%	13.0%	20.2%	20.0%
Logistic Regression	40.0%	40.4%	51.0%	51.0%
Human	80.3%	77.0%	90.5%	86.8%

Table 5: Performance of various methods and humans. Logistic regression outperforms the baselines, while there is still a significant gap between humans.

Train-Test Splits

- What we really care about is performance on new data we haven't seen before
  - I.e., data the model hasn't trained on
- We need to simulate this scenario
- We "hold out" some data from training, so model can't use it to set parameters
- Then we evaluate on the held out data

Train Test Splits

Beyond IID

- i.i.d.: train and test data are drawn from the same distribution
  - I.e., take a dataset, randomly shuffle it, and split it into 80% train/20% test
  - This is the most standard setting, a “traditional ML” setting
- In real applications, test isn’t always i.i.d., i.e.,
  - You want to build a model of customers that generalizes to new markets (train in China, test in US)
  - You want to forecast disease spread that generalizes to the future (train in 2010—2019, test in 2020—2021)
  - You want to screen applicants for an internship, based on data on success of past interns (train data is from 1980—2000 when company was primarily white upper middle class, new applicant pool is more racially and socio-economically diverse)

Train Test Splits

Practice Question!

- Context: Social media director for a PR company
- Data: Instagram posts for 5000 new musicians, plus subsequent likes/reposts/comments and sales records.
- Goal: Predict the popularity of a post so we can optimize visibility of new clients.

How should I define my train/test splits here?

a) i.i.d., i.e., randomly split posts into train/test
b) hold out posts from the most recent year
c) hold out posts from 10% of artists
d) hold out least popular 10% of posts

Train Test Splits

Practice Question!

- Context: Social media director for a PR company
- Data: Instagram posts for 5000 new musicians, plus subsequent likes/reposts/comments and sales records.
- Goal: Predict the popularity of a post so we can **optimize visibility of new clients**.

How should I define my train/test splits here?

a) i.i.d., i.e., randomly split posts into train/test
b) hold out posts from the most recent year
c) hold out posts from 10% of artists
d) hold out least popular 10% of posts

Train Test Splits

Practice Question!

- Context: Hedge fund manager
- Data: Social media chatter about companies plus daily stock prices for those companies over past .
- Goal: Detect when there is going to be another “GameStop situation”…i.e., sudden spike in a stoke’s price

How should I define my train/test splits here?

a) i.i.d.: randomly split daily returns into train/test
b) hold out data from the most recent N years
c) hold out data from 10% of companies
d) hold out data from companies that experienced spikes

Train Test Splits

Practice Question!

- Context: Hedge fund manager
- Data: Social media chatter about companies plus daily stock prices for those companies over past .
- Goal: **Detect when** there is going to be another “GameStop situation”…i.e., sudden spike in a stoke’s price

How should I define my train/test splits here?

a) i.i.d.: randomly split daily returns into train/test
b) hold out data from the most recent N years
c) hold out data from 10% of companies
d) hold out data from companies that experienced spikes

Topics

- Train-Test Splits and Baselines
- **Preprocessing**
- **Feature Engineering**
  - Weighting Strategies
  - Ngrams
  - More Advanced Features

Preprocessing vs. Features

- No clear line!
- Generally:
  - **Preprocessing** steps are about defining the **vocabulary**. Decisions about preprocessing affect all the features.
  - **Features** are about capturing aspects of the language that will help the model solve the task. **Can be as complex as you want** them to be, and each feature is independent.

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

plain text documents (e.g., scraped from the internet)

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

"cleaned up" text, often in some simple data structure (list of words).

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

"cleaned up" text, often in some simple data structure (list of words).

(sometimes more complex, e.g., if your features require some metadata)

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

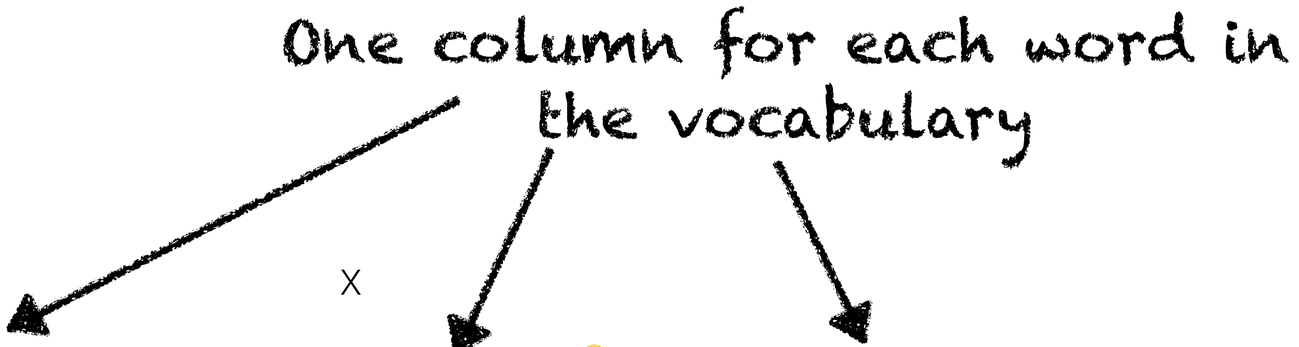
```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

feature matrix
(n_examples x n_features
matrix with numeric values)

Topics

- Train-Test Splits and Baselines
- **Preprocessing**
- Feature Engineering
  - Weighting Strategies
  - Ngrams
  - More Advanced Features

Preprocessing



	X										
Y	New	Tax	This	600	baby	…	👇	18	healthy	foods	that
0	1	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Preprocessing

"word" and "vocabulary" are both up to you!

	X										
Y	New	Tax	This	600	baby	…	👇	18	healthy	foods	that
0	1	1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Preprocessing

Are these the same word?

- “color” vs. “colour”?
- “apple” vs. “Apple”?
- “laugh” vs. “laughing” vs. “laughter”?
- “so” vs. “soooooooooooooo”?
- “laugh” vs. “giggle”?
- “apple” vs. “fruit”?

Preprocessing

Are these the same word?

maybe yes if you are building QA system

probably no if you are building a geo-locator

- “color” vs. “colour”?
- “apple” vs. “Apple”?
- “laugh” vs. “laughing” vs. “laughter”?
- “so” vs. “soooooooooooooo”?
- “laugh” vs. “giggle”?
- “apple” vs. “fruit”?

Preprocessing

Are these the same word?

- “color” vs. “colour”?
- “apple” vs. “Apple”?
- “laugh” vs. “laughing” vs. “laughter”?
- “so” vs. “soooooooooooooo”?
- “laugh” vs. “giggle”?
- “apple” vs. “fruit”?

maybe yes if you are using social media/chat data

probably no if you are working with WSJ data

Preprocessing

Are these the same word?

- “color” vs. “colour”?
- “apple” vs. “Apple”?
- “laugh” vs. “laughing” vs. “laughter”?
- “so” vs. “soooooooooooooo”?
- “laugh” vs. “giggle”?
- “apple” vs. “fruit”?

maybe yes if you are building and IR system

probably no if you are training a speech recognition system

Preprocessing

Are these the same word?

- “color” vs. “colour”?
- “apple” vs. “Apple”?
- “laugh” vs. “laughing” vs. “laughter”?
- “so” vs. “sooooooooooooooo”?
- “laugh” vs. “giggle”?
- “apple” vs. “fruit”?

maybe yes if you are building an ad-matching model

probably no if you are doing machine translation

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

Less common, but worth knowing about

Prep

Comm

Prep

Comm

Three basic levers of control: </h2> 1. Grind Without controversy, grind is the most important element. Without a consistent grind, nothing else you do will be able to correct how your coffee extracts. You cannot dose, distribute, tamp or pull your way out of a bad grind. - Using a burr grinder is key for a consistent grind size when making espresso. You cannot use a chop grinder and achieve the needed grind consistency <ul class="ul2"><ul class="ul2"><li class="li6">Chopping your beans with a tiny spinning blade will leave you with messy variables within your grind, causing your coffee to extract poorly; over-extracting the finer grounds and under extracting the coarser grounds.
<ul class="ul2"><ul class="ul2"><li class="li6">Dial in the SAI Millwright Hand Grinder for the perfect espresso grind. 2. Dose Your dose is how much ground coffee you put in your portafilter basket before distributing and tamping. - It is important to Dose by the volumetric standard of your portafilter basket. - After Dosing, evenly distribute your grounds with the SAI BT distribution tool, and tamp uniformly with the SAI New Levy tool.

Preprocessing

Common Steps

- **Strip boilerplate/html/etc**
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

Preprocessing

Stripping Boilerplate

Three basic levers of control: </h2> 1. Grind Without controversy, grind is the most important element. Without a consistent grind, nothing else you do will be able to correct how your coffee extracts. You cannot dose, distribute, tamp or pull your way out of a bad grind. - Using a burr grinder is key for a consistent grind size when making espresso. You cannot use a chop grinder and achieve the needed grind consistency <ul class="ul2"><ul class="ul2"><li class="li6">Chopping your beans with a tiny spinning blade will leave you with messy variables within your grind, causing your coffee to extract poorly; over-extracting the finer grounds and under extracting the coarser grounds.
<ul class="ul2"><ul class="ul2"><li class="li6">Dial in the SAI Millwright Hand Grinder for the perfect espresso grind. 2. Dose Your dose is how much ground coffee you put in your portafilter basket before distributing and tamping. - It is important to Dose by the volumetric standard of your portafilter basket. - After Dosing, evenly distribute your grounds with the SAI BT distribution tool, and tamp uniformly with the SAI New Levy tool.

Preprocessing

Stripping Boilerplate

- Typically done with regex
- There are libraries for standard things (html)
  - beautifulsoup is great for html
- Can be dataset specific
  - e.g., “Madam Speaker…”

Three basic levers of control: 1. Grind Without controversy, grind is the most important element.Without a consistent grind, nothing else you do will be able to correct how your coffee extracts. You cannot dose, distribute, tamp or pull your way out of a bad grind.- Using a burr grinder is key for a consistent grind size when making espresso. You cannot use a chop grinder and achieve the needed grind consistencyChopping your beans with a tiny spinning blade will leave you with messy variables within your grind, causing your coffee to extract poorly; over-extracting the finer grounds and under extracting the coarser grounds.Dial in the SAI Millwright Hand Grinder for the perfect espresso grind. 2. Dose Your dose is how much ground coffee you put in your portafilter basket before distributing and tamping.- It is important to Dose by the volumetric standard of your portafilter basket.- After Dosing, evenly distribute your grounds with the SAI BT distribution tool, and tamp uniformly with the SAI New Levy tool.

Preprocessing

Stripping Boilerplate

- Can choose to hand-write rules to clean up further, but these can be page specific and might not help performance
- Real data will always have some weirdness to it

Three basic levers of control: 1. Grind Without controversy, grind is the most important element.Without a consistent grind, nothing else you do will be able to correct how your coffee extracts. You cannot dose, distribute, tamp or pull your way out of a bad grind.- Using a burr grinder is key for a consistent grind size when making espresso. You cannot use a chop grinder and achieve the needed grind consistencyChopping your beans with a tiny spinning blade will leave you with messy variables within your grind, causing your coffee to extract poorly; over-extracting the finer grounds and under extracting the coarser grounds.Dial in the SAI Millwright Hand Grinder for the perfect espresso grind. 2. Dose Your dose is how much ground coffee you put in your portafilter basket before distributing and tamping.- It is important to Dose by the volumetric standard of your portafilter basket.- After Dosing, evenly distribute your grounds with the SAI BT distribution tool, and tamp uniformly with the SAI New Levy tool.

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- **Tokenization**
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

Preprocessing

Tokenization

- Need to split into “words”!
- Often, split on whitespace is the simplest way
  - Works okay for English, but not other languages

[‘1.’, ‘Grind’, ‘Without’, ‘controversy,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts.’, ‘You’, ‘cannot’, ‘dose,’, ‘distribute,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind.-’]

Preprocessing

Tokenization

- Need to split into “words”!
- Often, split on whitespace is the simplest way
  - Works okay for English, but not other languages

[‘1.’, ‘Grind’, ‘Without’, ‘controversy,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts.’, ‘You’, ‘cannot’, ‘dose,’, ‘distribute,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind.-’]

Preprocessing

Tokenization

- Need to split into “words”!
- Often, split on whitespace is the simplest way

X

Grind	grind,	grind.-	element. Without	…
1	1	0	0	0
0	0	1	1	1
0	0	1	0	0

[‘1.’, ‘Grind’, ‘Without’, ‘controversy,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts.’, ‘You’, ‘cannot’, ‘dose,’, ‘distribute,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind.-’]

Preprocessing

Tokenization

- Need to split into “words”!
- Often, split on whitespace is the simplest way
  - Works for English, but not other languages
- Usually, use tokenization models that, e.g., split off punctuation (more next lecture)

[‘1’ ‘.’, ‘Grind’, ‘Without’, ‘controversy’, ‘,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts’, ‘.’, ‘You’, ‘cannot’, ‘dose’, ‘,’, ‘distribute’, ‘,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind’, ‘.’, ‘-’]

Preprocessing

Tokenization

- Need to split into “words”!
- Often, split on whitespace is the simplest way

X

Grind	grind	element.Without	…
1	1	0	0
0	0	1	1
0	0	0	0

feature)

[‘1’ ‘.’, ‘Grind’, ‘Without’, ‘controversy’, ‘,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts’, ‘.’, ‘You’, ‘cannot’, ‘dose’, ‘,’, ‘distribute’, ‘,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind’, ‘.’, ‘-’]

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- **Stemming/Lemmatization**
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

Preprocessing

Stemming/Lemmatization

- We want different forms of a word to count as the same!
  - strips plural markers (“dogs” -> dog)
  - strips verb conjugations (-ing, -ed, etc)
- Custom models for stripping off “extra” info (more next lecture)

[‘1’ ‘.’, ‘Grind’, ‘Without’, ‘controversy’, ‘,’, ‘grind’, ‘is’, ‘the’, ‘most’, ‘important’, ‘element.Without’, ‘a’ ‘consistent’, ‘grind,’, ‘nothing’, ‘else’, ‘you’, ‘do’, ‘will’, ‘be’, ‘able’, ‘to’, ‘correct’, ‘how’, ‘your’, ‘coffee’, ‘extracts’, ‘.’, ‘You’, ‘cannot’, ‘dose’, ‘,’, ‘distribute’, ‘,’, ‘tamp’, ‘or’, ‘pull’, ‘your’, ‘way’, ‘out’, ‘of’, ‘a’, ‘bad’, ‘grind’, ‘.’, ‘-’]

Preprocessing

Stemming/Lemmatization

- We want different forms of a word to count as the same!
  - strips plural markers ("dogs" -> dog)
  - strips verb conjugations (-ing, -ed, etc)
- Custom models for stripping off "extra" info (more next lecture)

['1' '.', 'Grind', 'Without', 'controversy', ',', 'grind', 'be', 'the', 'most', 'important', 'element.Without', 'a' 'consistent', 'grind,', 'nothing', 'else', 'you', 'do', 'will', 'be', 'able', 'to', 'correct', 'how', 'your', 'coffee', 'extract', '.', 'You', 'cannot', 'dose', ',', 'distribute', ',', 'tamp', 'or', 'pull', 'your', 'way', 'out', 'of', 'a', 'bad', 'grind', '.', '-']

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- **Lowercasing**
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

Preprocessing

Stemming/Lemmatization

- Usually we lowercase everything
- Typically happens after other preprocessing steps (e.g., parsing/syntax) since case can be a good feature
- In some applications, lowercase hurts performance!

['1' '.', 'Grind', 'Without', 'controversy', ',', 'grind', 'be', 'the', 'most', 'important', 'element.Without', 'a' 'consistent', 'grind,', 'nothing', 'else', 'you', 'do', 'will', 'be', 'able', 'to', 'correct', 'how', 'your', 'coffee', 'extract', '.', 'You', 'cannot', 'dose', ',', 'distribute', ',', 'tamp', 'or', 'pull', 'your', 'way', 'out', 'of', 'a', 'bad', 'grind', '.', '-']

Preprocessing

Stemming/Lemmatization

- Usually we lowercase everything
- Typically happens after other preprocessing steps (e.g., parsing/syntax) since case can be a good feature
- In some applications, lowercase hurts performance!

['1' '.', 'grind', 'without', 'controversy', ',', 'grind', 'be', 'the', 'most', 'important', 'element.without', 'a' 'consistent', 'grind,', 'nothing', 'else', 'you', 'do', 'will', 'be', 'able', 'to', 'correct', 'how', 'your', 'coffee', 'extract', '.', 'you', 'cannot', 'dose', ',', 'distribute', ',', 'tamp', 'or', 'pull', 'your', 'way', 'out', 'of', 'a', 'bad', 'grind', '.', '-']

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- **Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)**
- Setting a Vocab Size/Defining “<OOV>”
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

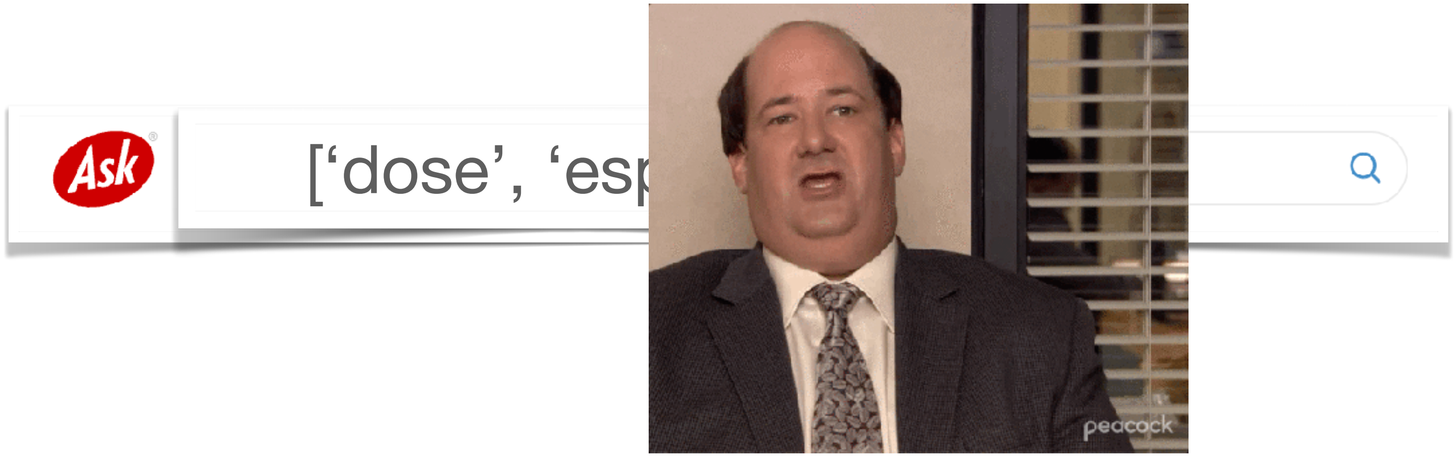
Preprocessing

Removing Stop Words

- Lots of (very cool! very important! very special!) words are unlikely to make a difference for tasks
  - Punctuation
  - "Stop Words" (top K in vocab, or hand-picked)
  - Numbers (too sparse)

['1' '.', 'grind', 'without', 'controversy', ',', 'grind', 'be', 'the', 'most', 'important', 'element.without', 'a' 'consistent', 'grind,', 'nothing', 'else', 'you', 'do', 'will', 'be', 'able', 'to', 'correct', 'how', 'your', 'coffee', 'extract', '.', 'you', 'cannot', 'dose', ',', 'distribute', ',', 'tamp', 'or', 'pull', 'your', 'way', 'out', 'of', 'a', 'bad', 'grind', '.', '-']

Preprocessing
Removing Stop Words



- Lots of (very cool! very important! very special!) words are unlikely to make a difference for tasks
  - Punctuation
  - "Stop Words" (top K in vocab, or hand-picked)
  - Numbers (too sparse)

['<NUM>', 'grind', 'without', 'controversy', 'grind', 'most', 'important', 'element.without', 'consistent', 'grind,', 'nothing', 'else', 'able', 'correct', 'coffee', 'extract', 'cannot', 'dose', 'distribute', 'tamp', 'pull', 'way', 'out', 'bad', 'grind']

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- **Setting a Vocab Size/Defining “<OOV>”**
- Word Sense Disambiguation
- Combining Synonyms/Paraphrases

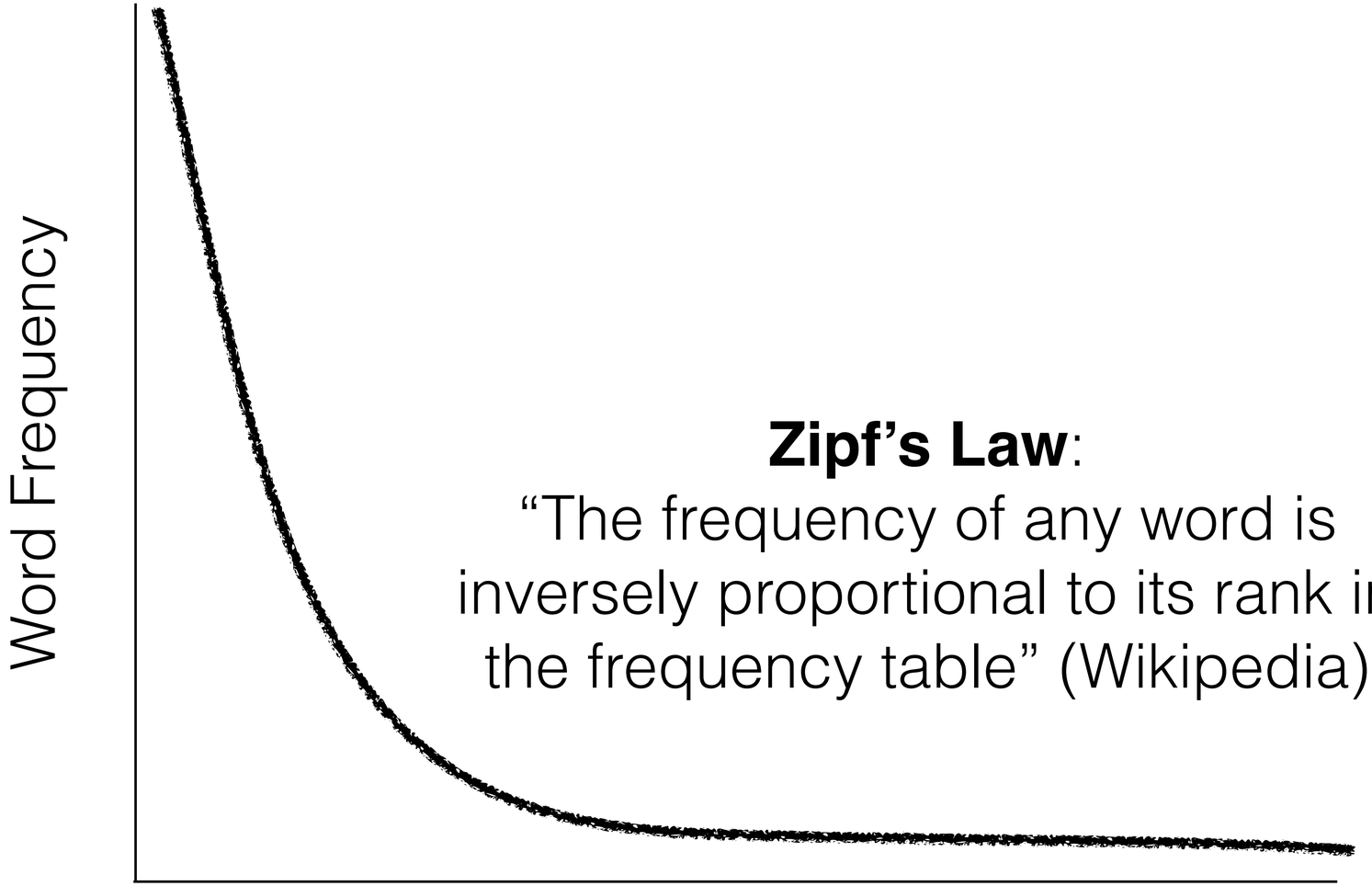
Preprocessing

Vocab Size and OOV

- Natural language vocabularies are huge!
  - (Even more so when we have noisy data, misspellings, tokenization errors, etc)
- Most words occur very rarely
- Even if these are “good” or “interesting” words, they are unlikely to be much help in practice

Preprocessing

Vocab Size and OOV

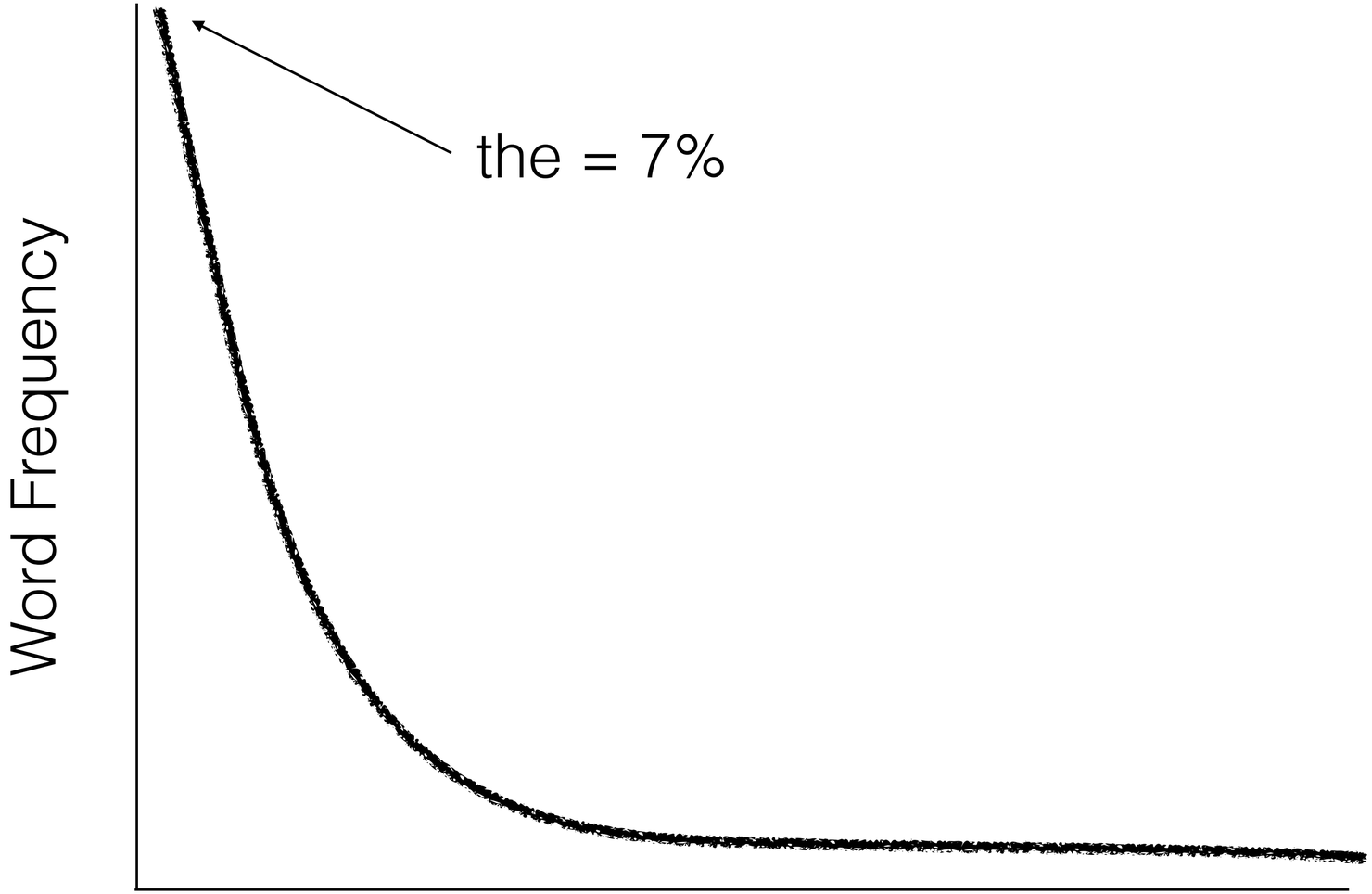


Zipf's Law:
"The frequency of any word is inversely proportional to its rank in the frequency table" (Wikipedia)

https://en.wikipedia.org/wiki/Zipf%27s_law

Preprocessing

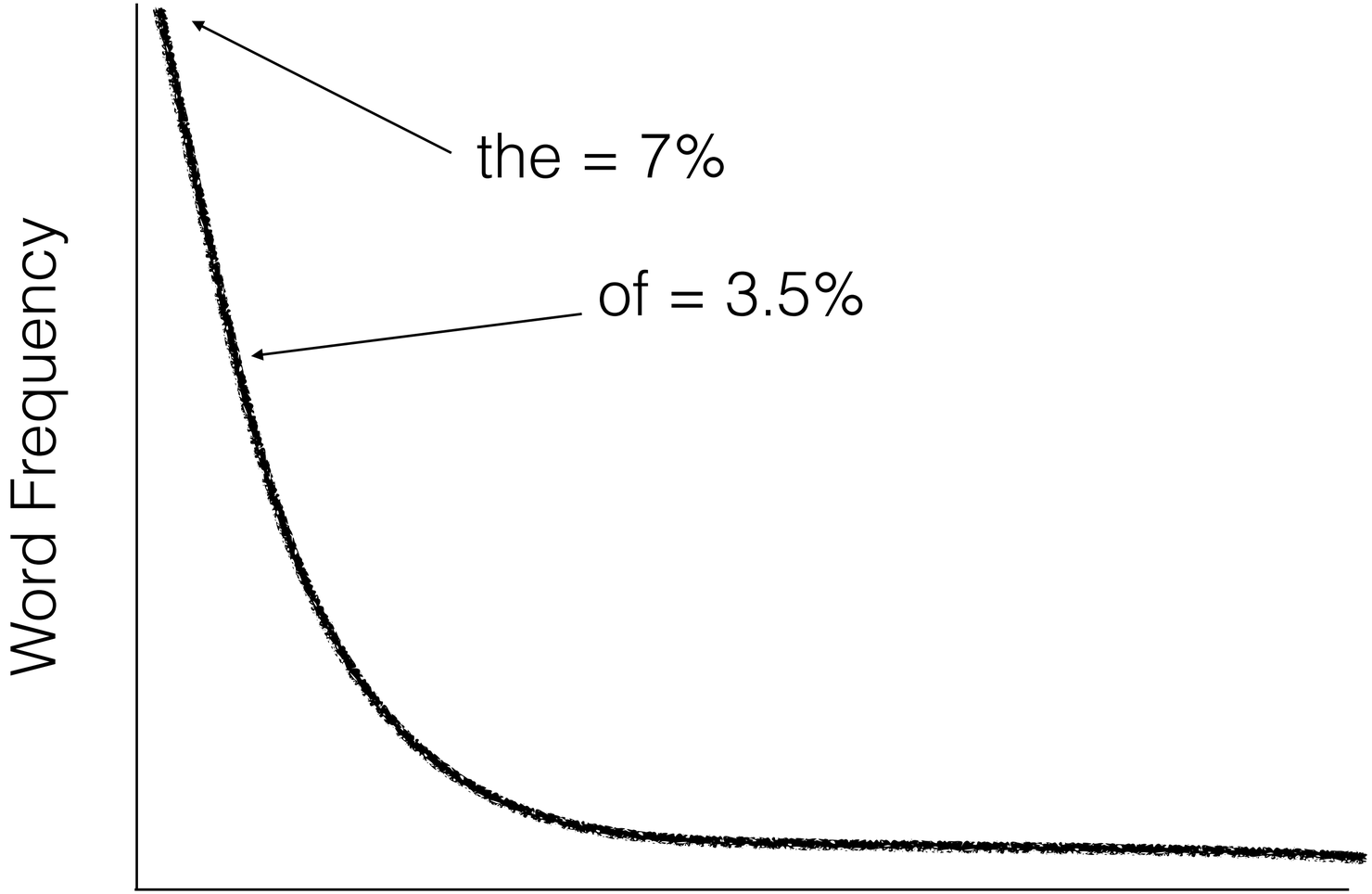
Vocab Size and OOV



https://en.wikipedia.org/wiki/Zipf%27s_law

Preprocessing

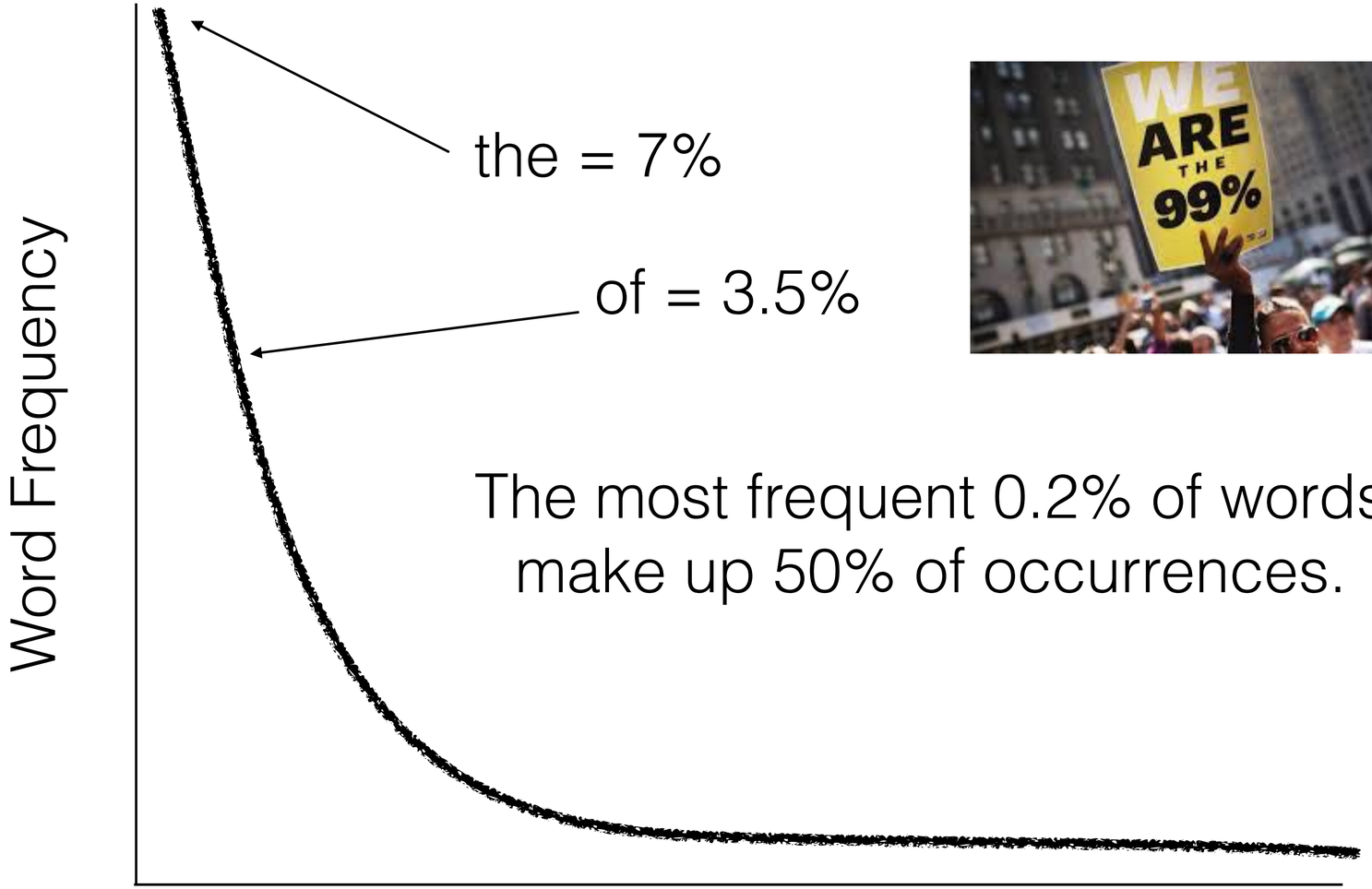
Vocab Size and OOV



https://en.wikipedia.org/wiki/Zipf%27s_law

Preprocessing

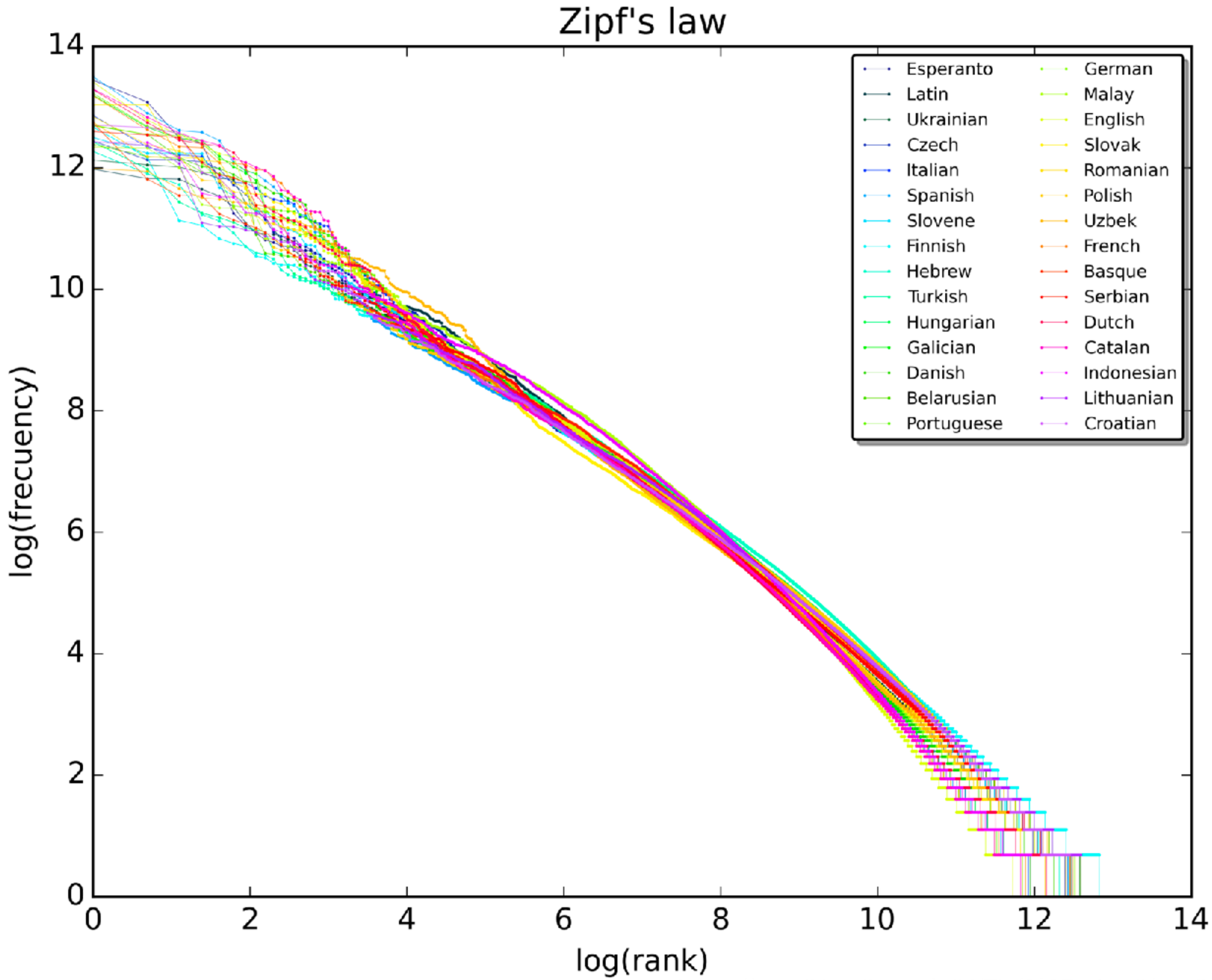
Vocab Size and OOV



The most frequent 0.2% of words make up 50% of occurrences.

https://en.wikipedia.org/wiki/Zipf%27s_law

Pre
Vocal



https://en.wikipedia.org/wiki/Zipf%27s_law#/media/File:Zipf_30wiki_en_labels.png

Preprocessing
Vocab Size and OOV

- Typically, set some size N (e.g., 20K) and keep only the most frequent N words
- Alternatively, set a threshold k (e.g., 10) and throw away words that occur fewer than k times
- Replace others with special token (<OOV> or <UNK>)

['<NUM>', 'grind', 'without', 'controversy', 'grind', 'most', 'important', 'element.without', 'consistent', 'grind,', 'nothing', 'else', 'able', 'correct', 'coffee', 'extract', 'cannot', 'dose', 'distribute', 'tamp', 'pull', 'way', 'out', 'bad', 'grind']

Preprocessing
Vocab Size and OOV

- Typically, set some size N (e.g., 20K) and keep only the most frequent N words
- Alternatively, set a threshold k (e.g., 10) and throw away words that occur fewer than k times
- Replace others with special token (<OOV> or <UNK>)

['<NUM>', 'grind', 'without', '<OOV>', 'grind', 'most', 'important', '<OOV>', '<OOV>', 'grind,', 'nothing', 'else', 'able', 'correct', 'coffee', 'extract', 'cannot', 'dose', 'distribute', 'tamp', 'pull', 'way', 'out', 'bad', 'grind']

Preprocessing

Common Steps

- Strip boilerplate/html/etc
- Tokenization
- Stemming/Lemmatization
- Lowercasing
- Removing “Semantically Vacuous” Words (Punctuation, Stop Words, Numbers)
- Setting a Vocab Size/Defining “<OOV>”
- **Word Sense Disambiguation**
- **Combining Synonyms/Paraphrases**

Preprocessing

Word Sense Disambiguation

- “homographs” -> string identical, but different “meanings”
  - sometimes called “polysemes”/“polysemy” in NLP
- E.g., “bat”, “bank”, “park”, “fly”, …
- Lots of research on:
  - Building inventories of word senses
  - Building models to identify, for a given instance of a word, which “sense” it is

Preprocessing

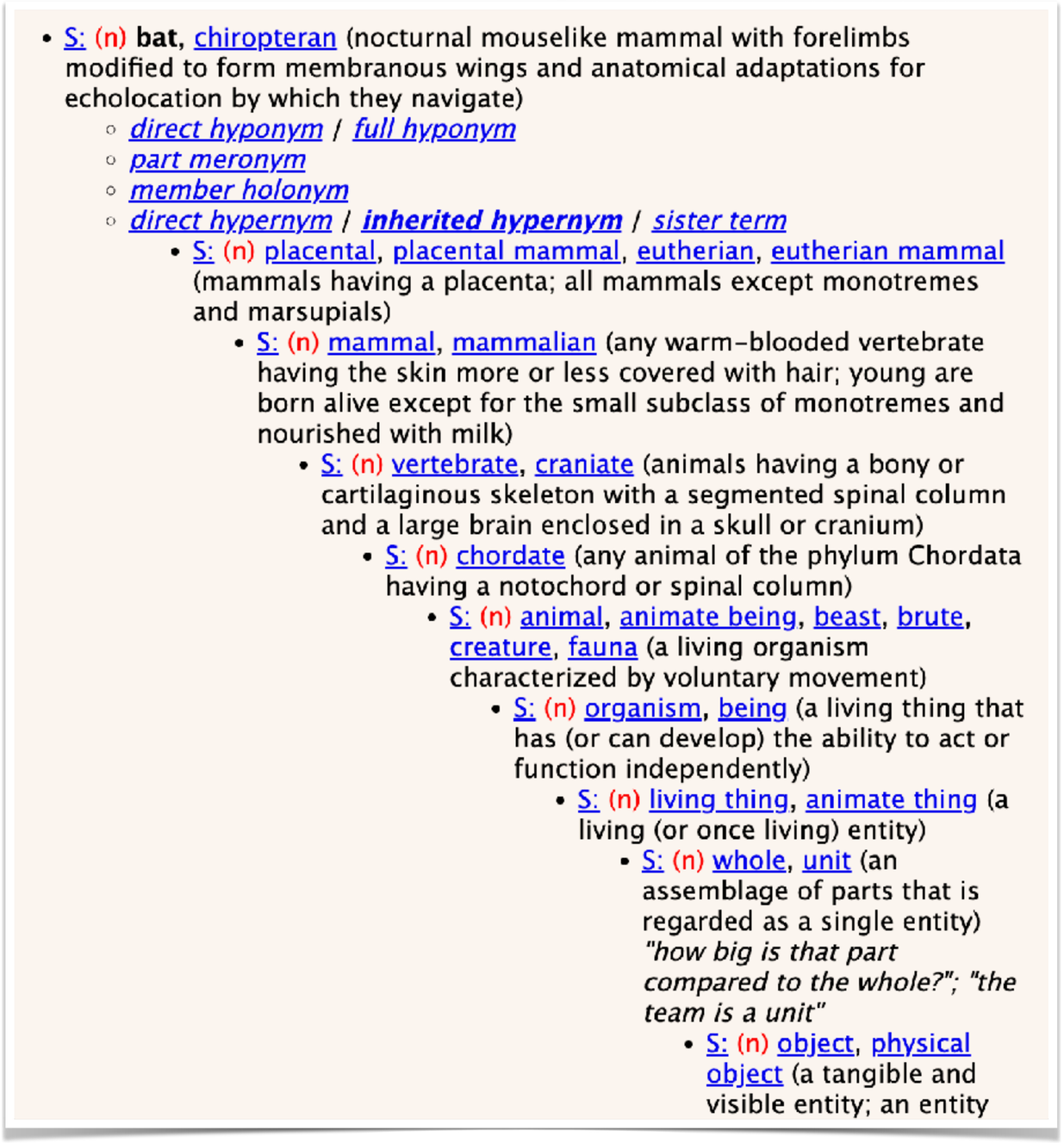
WordNet

- http://wordnetweb.princeton.edu/perl/webwn
- Organizes words into “synsets” (groups of words with the same meaning)
- Synsets are then organized hierarchically (nouns) or as a graph (verbs, adjectives)

Preprocessing

WordNet

- http://wordnetweb.princeton.edu/perl/webwn
- Organizes words into “synsets” (groups of words with the same meaning)
- Synsets are then organized hierarchically (nouns) or as a graph (verbs, adjectives)



Preprocessing

Lexical Paraphrase/Differentiation

- During preprocessing, some use tools like part of speech taggers, WSD, and/or WordNet to map words to more abstract categories
- You might see, e.g.,:
  - bat_NN -> bat as a noun (vs. verb)
  - bat.n.01 -> bat as a noun, specifically the first synset in WordNet
  - bat.n.01_animal.n.03_entity.n.01, … -> adding in additional information about *hypernym* info

['<NUM>', 'grind.n.02', 'without', '<OOV>', 'grind.n.02', 'most', 'important', '<OOV>', '<OOV>', 'grind.n.02,', 'nothing', 'else', 'able', 'correct', 'coffee', 'extract.n.04', 'cannot', 'dose.v.04', 'distribute', 'tamp.v.02', 'pull.v.08', 'way', 'out', 'bad', 'grind.n.02']

Preprocessing

Just for fun!

Preprocessing

Just for fun!

take dog get free cup <OOV> <OOV> . sit outside , course , love .
favorite <OOV> <OOV> <OOV> <OOV> waffle <OOV> . <OOV> thing try , feel
like time want try new . stick favorite :)

Preprocessing

Just for fun!

took our dog there and she got a free cup of vanilla gelato. we sat outside, of course, but she loved it. my favorite combo is coconut and roasted almond with a waffle cone. soooo many things to try, I feel like every time I go in I want to try something new. but I always stick with my favorite :)

Preprocessing

Just for fun!

<OOV> room <OOV> <OOV> , <OOV> <OOV> . food expensive . egg , slice toast , cup hash - $ <OOV> . server way <OOV> . stay <OOV> , like , stay away . day . <OOV> repair <OOV> <OOV> try <OOV> smell <OOV> <OOV> bathroom . let know <OOV> . , <OOV> <OOV> . room <OOV> . sure bad , <OOV> , <OOV> <OOV> ! case , bathroom <OOV> <OOV> <OOV> . maybe fix leave . bad " star " rating . understand comment <OOV> high price pay room . expectation high $ <OOV> .

Preprocessing

Just

There are no bathtubs in any rooms except the executive suites, only narrow showers. The food is expensive. Two eggs, two slices of toast, one cup of hash - $17. Servers are way too patronizing. Unless you have to stay here for a conference, like me, stay away. And this is only my first day. Had to call maintenance to repair a leaky toilet and to try and avert the smell of sewer gas in the bathroom. I will let you know if they corrected it. Well, the operator lied to me. There are rooms with bathtubs. not sure which is worse, the lie, or a narrow shower! In any case, my bathroom still stinks and the toilet still leaks. Maybe they will fix it after I leave. Too bad they don't have "no stars" as a rating. Understand that my comments are all related to the high price I am paying for the room. My expectations are as high as $200 or so.

Preprocessing

Just for fun!

food good spicy instead flavorful sauce salsa taste like tomato service bit <OOV> waiter actually leave <OOV> meal order vegetarian chili <OOV> serve meat atmosphere amazing beautiful restaurant <OOV> food <OOV> <OOV> location <OOV> perfect

Preprocessing

Just for fun!

Food was good, spicy instead of flavorful. All the sauces/salsas tasted too much like tomato to me. Service was a bit apathetic waiter actually left halfway through meal. Ordered a vegetarian chili relleno and was served one with meat. Atmosphere is amazing, beautiful restaurant. If you could meld the food of Poca Cosa with the location of Penca it would be perfect.

Preprocessing

Wrap Up

- “Preprocessing” refers to steps we take to “clean up” the language input before extracting features
- Preprocessing pipelines vary significantly from one use case to the next
- Researchers/engineers tend to try lots of things and see what works for their application and data
  - (Typically, use cross-validation on train/dev to decide on strategies. Then, at the end, test on a held out test!)

Topics

- Train-Test Splits and Baselines
- Preprocessing
- **Feature Engineering**
  - Weighting Strategies
  - Ngrams
  - More Advanced Features

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(raw_data)
X = extract_features(X)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

feature matrix
(n_examples x n_features
matrix with numeric values)

Preprocessing vs. Features

Disclaimer: This is pseudocode!

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(r
X = extract_features(preprocesse
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

```
def extract_features(dat):
  feature_fn1(dat)
  feature_fn2(dat)
  …
  feature_fnk(dat)
```

typically calls a series of functions, each computing a different feature/set of features. these functions can be simple or very complex.

Topics

- Train-Test Splits and Baselines
- Preprocessing
- Feature Engineering
  - **Weighting Strategies**
  - Ngrams
  - More Advanced Features

Preprocessing vs. Features

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(r
X = extract_features(preprocessed_data)
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

```
def extract_features(dat):
  get_bow_features(dat)
  …
```

Feature Engineering

Weighting BOW Models

- **Basic BOW**: 1 if a word is present, 0 otherwise
  - This works surprisingly well!
- **Count-Based**: # of times word appears in the document
- **tf-idf**: Weighting scheme designed to upweight “important” words

Feature Engineering

tf-idf

- Assigns higher weights to words that differentiate this document from other documents
- tf-idf(word,doc) = (# times word appears in doc) / (# of times word appears across all documents)
- Can filter out low tf-idf words or else just reweight the term-document matrix accordingly

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

doc 3

webdev: html does work

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

doc 3

webdev: html does work

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

What is the tf-idf vector for doc1?

a)

1/3	1/3	1	1/3	0	1/2	1	0	1

b)

1/2	1/3	1	1/3	1	1/2	0	1/2	1

c)

1/3	1/3	1	1/2	1	1/2	0	0	0

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

3

html
work

df

html: 3
does: 3
not: 1
work: 2
at: 1
all: 2
webdev: 2
is: 1
awesome: 1

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

What is the tf-idf vector for doc1?

a)

1/3	1/3	1	1/3	0	1/2	1	0	1

b)

1/2	1/3	1	1/3	1	1/2	0	1/2	1

c)

1/3	1/3	1	1/2	1	1/2	0	0	0

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

3

html
work

df

html: 3
does: 3
not: 1
work: 2
at: 1
all: 2
webdev: 2
is: 1
awesome: 1

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

What is the tf-idf vector for doc1?

a)

1/3	1/3	1	1/3	0	1/2	1	0	1

b)

1/2	1/3	1	1/3	1	1/2	0	1/2	1

c)

1/3	1/3	1	1/2	1	1/2	0	0	0

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

3

html
work

df
html: 3
does: 3
not: 1
work: 2
at: 1
all: 2
webdev: 2
is: 1
awesome: 1

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

What is the tf-idf vector for doc1?

a)

1/3	1/3	1	1/3	0	1/2	1	0	1

b)

1/2	1/3	1	1/3	1	1/2	0	1/2	1

c)

1/3	1/3	1	1/2	1	1/2	0	0	0

Feature Engineering

tf-idf

doc1

html does not work

doc 2

html does work. all webdev is awesome.

3

html
work

df
html: 3
does: 3
not: 1
work: 2
at: 1
all: 2
webdev: 2
is: 1
awesome: 1

	html	does	not	work	at	all	webdev	is	awesome
doc1	1	1	1	1	1	1	0	0	0
doc 2	1	1	0	0	0	1	1	1	1
doc 3	1	1	0	1	0	0	1	0	0

What is the tf-idf vector for doc1?

a)

1/3	1/3	1	1/3	0	1/2	1	0	1

b)

1/2	1/3	1	1/3	1	1/2	0	1/2	1

c)

1/3	1/3	1	1/2	1	1/2	0	0	0

Topics

- Train-Test Splits and Baselines
- Preprocessing
- Feature Engineering
  - Weighting Strategies
  - **Ngrams**
  - More Advanced Features

Feature Engineering

N-Grams

- n-gram: sequence of words of length n
  - unigram: word
  - bigram: two-word phrase
  - trigram: three-word phrase
  - 4-gram: four-word phrase…
- Simple way to add information about “context” to a BOW model

Feature Engineering

N-Grams

```
# build up an n-gram vocabulary
for review in reviews:
  for i in (0, len(review)-n):
     vocab.add(review[i:i+n])

# for each review, include all ngrams as features,
# as though each one is a single word
for i in (0, len(review)):
  for j in (1,n):
     features.add(review[i:i+j])
```

Feature Engineering

N-Grams

The restaurant is not good. Go anywhere else!

The restaurant is good! Do not go anywhere else.

the	is	do	go	not	else	anywhere	restaurant	good
1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Feature Engineering

N-Grams

The restaurant is not good. Go anywhere else!

The restaurant is good! Do not go anywhere else.

the	is	do	go	not	else	anywhere	restaurant	good
1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Feature Engineering

N-Grams

The restaurant is not good. Go anywhere else!

The restaurant is good! Do not go anywhere else.

the	is	do	go	not	else	anywhere	restaurant	good	the restaurant	restaurant is	is not	is good	not good	good do	do not	good go	go anywhe re	anywhe re else	not go
1	1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	1	1

Feature Engineering

N-Grams

Disclaimer: This is pseudocode!

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(r
X = extract_features(preprocesse
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

```
def extract_features(dat):
  get_bow_features(dat)
  for i in i..5:
    get_ngrams(dat, i)
  …
```

Feature Engineering

N-Grams

- Assuming a vocabulary of size 10K, how many **unique bigrams** are there (worst case)?
  - 10K x 10K = 100M
- How many unique k-grams?
  - 10K^k!!
- In practice, these numbers are much much much lower.
  - Language is not random! Most ngrams are syntactically or semantically “illegal”, and will occur rarely or never
    - e.g., “the of”, “my the and”, “a tallest person”
- But its still a very large feature space!

Feature Engineering

N-Grams

- What is the right size of ngrams?
- It depends!
- Tradeoff between expressivity and generalization
- more ngrams = much larger feature matrix (most of which is 0), more parameters to estimate, more likely to overfit

Topics

- Train-Test Splits and Baselines
- Preprocessing
- Feature Engineering
  - Weighting Strategies
  - Ngrams
  - **More Advanced Features**

Feature Engineering

More Advanced Features

Disclaimer: This is pseudocode!

```
y, raw_data = load_data(file)
preprocessed_data = preprocess(r
X = extract_features(preprocesse
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

```
def extract_features(dat):
  get_bow_features(dat)
  for i in i..5:
    get_ngrams(dat, i)
  get_dependency_features(dat)
```

Feature Engineering

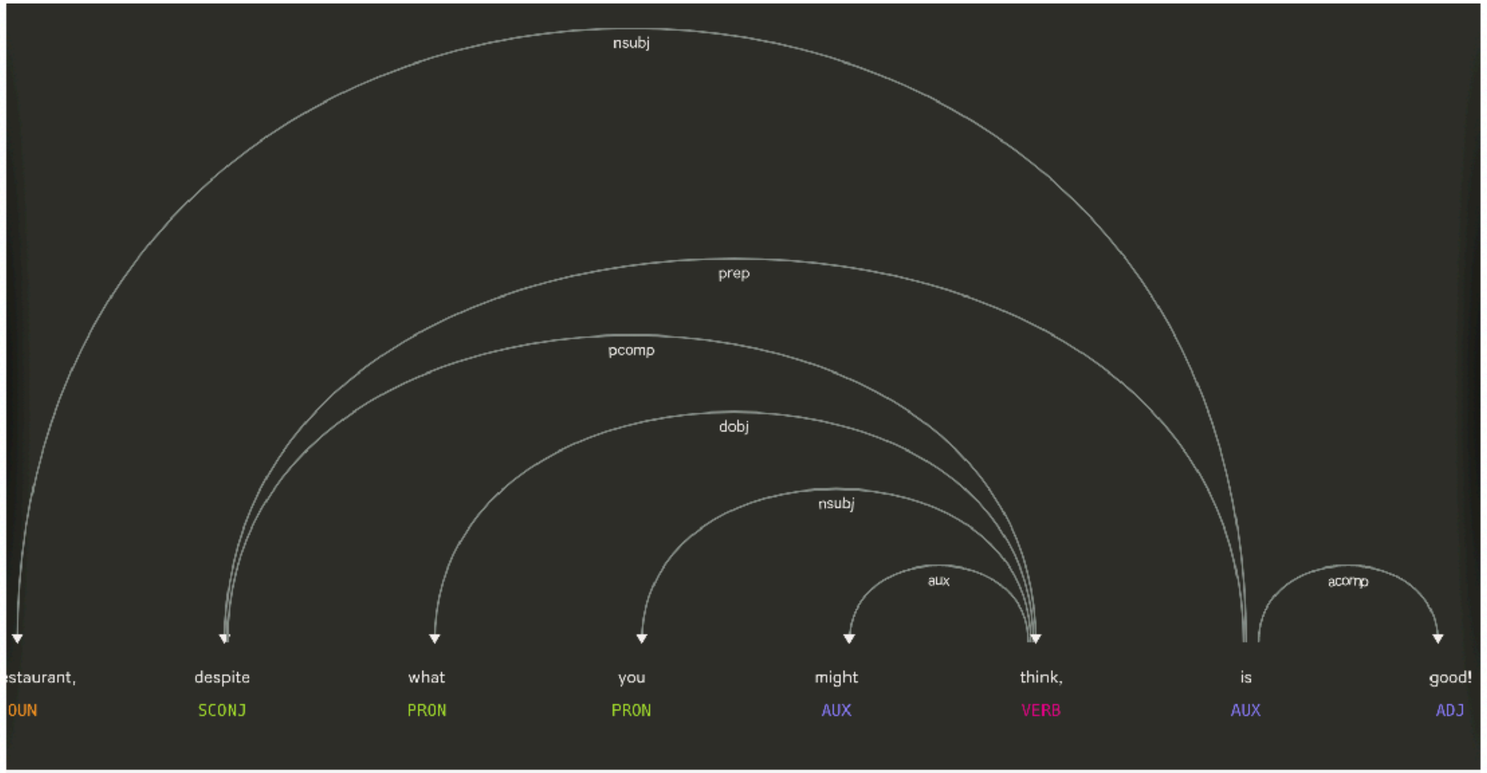
Syntactic Features

- Ngrams just capture physical proximity of words. Language is more flexible than that.
  - The restaurant is good!
  - The restaurant, with its adventurous nose-to-tail concept and avante garde reimagining of the use of salt, is, despite what you might think, good!

Feature Engineering

Syntactic Features

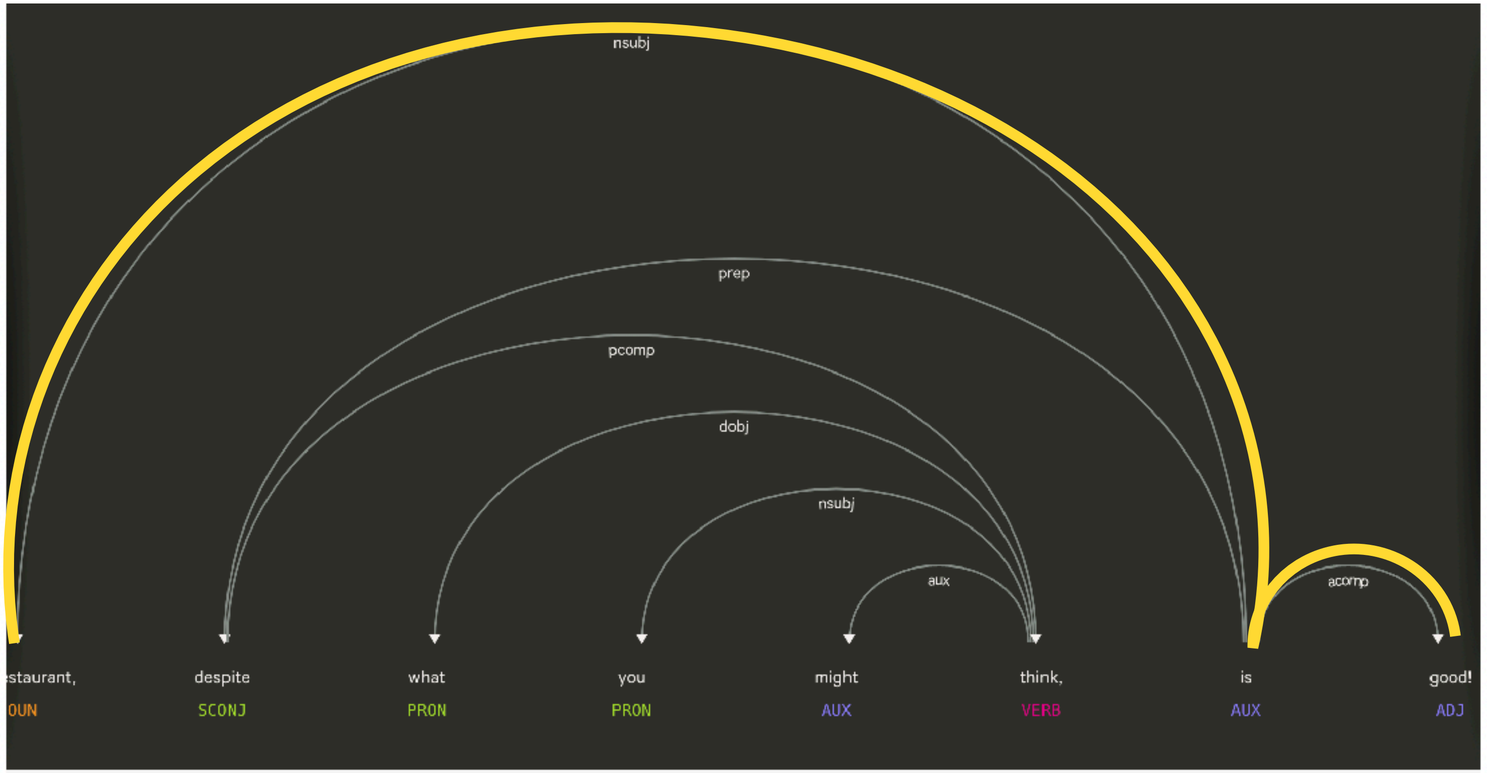
- People often use dependency parsers to find meaningful links between non-adjacent words



Feature Engineering

Syntactic Features

- People often use dependency parsers to find meaningful links between non-adjacent words



```
import spacy
import textwrap

def pretty_print(s):
  print("\n".join(textwrap.wrap(s)))

data = [l.strip().split('\t') for l in open('yelp_dataset_5K.txt').readlines()]
raw_data = [review for review, label in data]
y = [float(label) for review, label in data]

pretty_print(raw_data[1460])
```

```
mmmmmm. We just finished our first meal at this wonderful place (and
by finished I mean until we eat the rest for lunch tomorrow). Best
deep dish this side of the south side..... what a great crust, buttery
and crunchy even on the bottom. Well worth the wait for a fresh deep
dish chicago style! Why, oh why did we wait so long to hear of this
seminole heights jewel?! Only other place slightly close to this
outside of Chicago is Joey Ds and that is 45 minutes those of us in
Tampa dont meed to drive!!! Great service, quaint decor....YUM!!!! We
will be back! Take out and cash only... no delivery no CCs. Who
cares!!!
```

```
spacy_processor = spacy.load("en_core_web_sm")

parsed = spacy_processor(raw_data[1460])
for token in parsed:
  print(token.text, token.pos_)
```

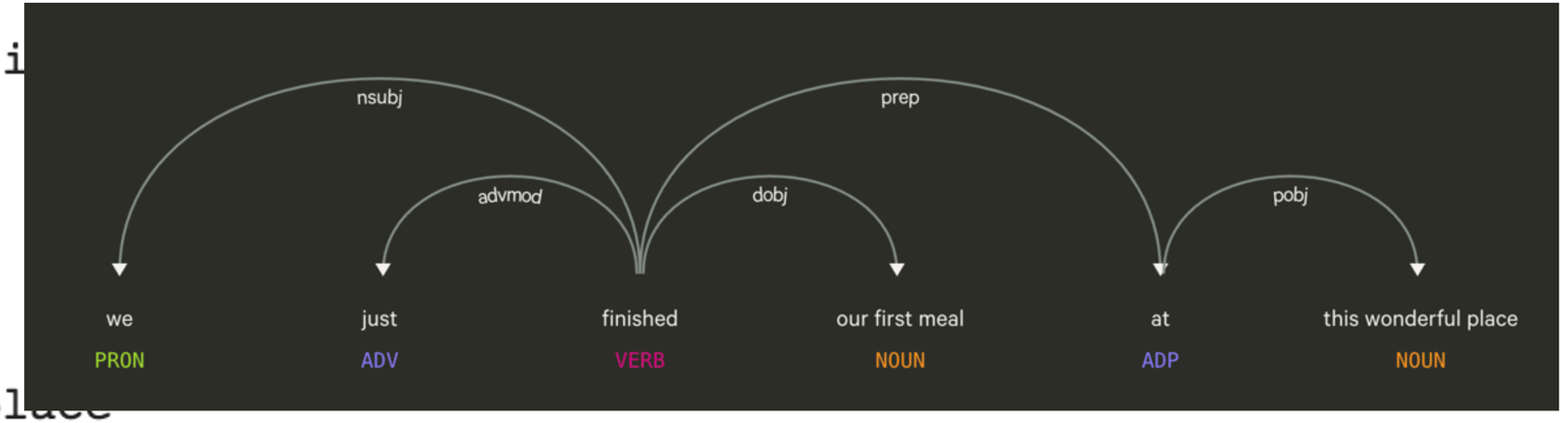
```
mmmmmm PROPN
. PUNCT
We PRON
just ADV
finished VERB
our PRON
first ADJ
meal NOUN
at ADP
this DET
wonderful ADJ
place NOUN
```

mmmmmm. We just finished our first meal at this wonderful place (and by finished I mean until we eat the rest for lunch tomorrow). Best deep dish this side of the south side..... what a great crust, buttery and crunchy even on the bottom. Well worth the wait for a fresh deep dish chicago style! Why, oh why did we wait so long to hear of this seminole heights jewel?! Only other place slightly close to this outside of Chicago is Joey Ds and that is 45 minutes those of us in Tampa dont meed to drive!!! Great service, quaint decor....YUM!!!! We will be back! Take out and cash only... no delivery no CCs. Who cares!!!

```python
spacy_processor = spacy.load("en_core_web_sm")

parsed = spacy_processor(raw_data[1460])
for token in parsed:
  print('\t'.join([token.dep_, token.text, token.head.text]))
```

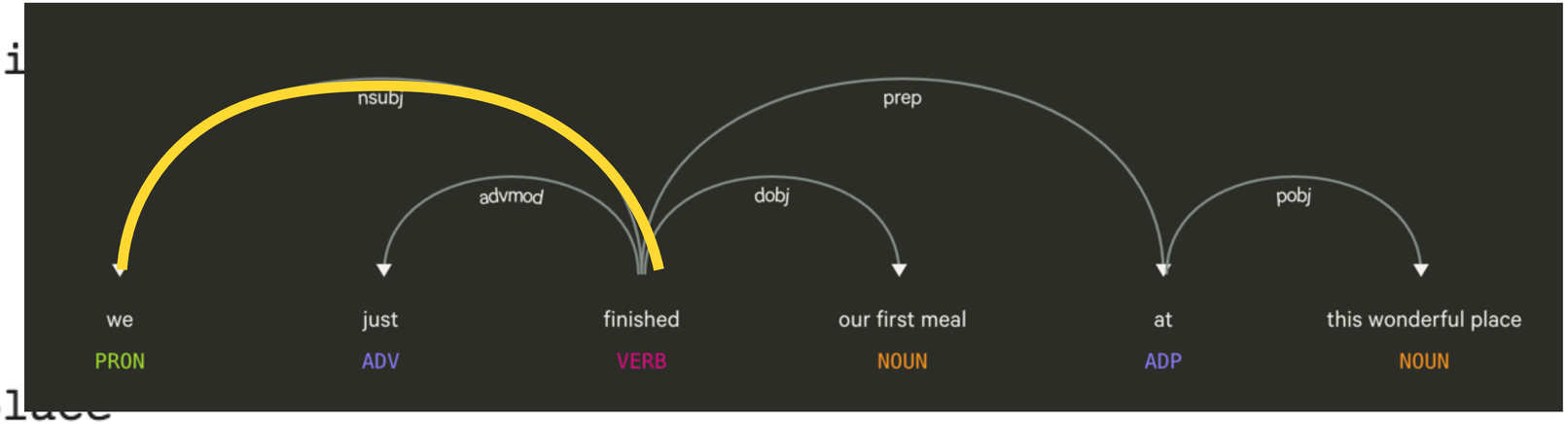
```
ROOT    mmmmmm  mmmmmm
punct   .       mmmmmm
nsubj   We      finished
advmod  just    finished
ROOT    finished        fi
poss    our     meal
amod    first   meal
dobj    meal    finished
prep    at      finished
det     this    place
amod    wonderful       place
pobj    place   at
punct   (       finished
cc      and     finished
conj    by      finished
```



```
spacy_processor = spacy.load("en_core_web_sm")

parsed = spacy_processor(raw_data[1460])
for token in parsed:
  print('\t'.join([token.dep_, token.text, token.head.text]))
```

```
ROOT    mmmmmm  mmmmmm
punct   .       mmmmmm
nsubj   We      finished
advmod  just    finished
ROOT    finished        fi
poss    our     meal
amod    first   meal
dobj    meal    finished
prep    at      finished
det     this    place
amod    wonderful       place
pobj    place   at
punct   (       finished
cc      and     finished
conj    by      finished
```



```
spacy_processor = spacy.load("en_core_web_sm")

parsed = spacy_processor(raw_data[1460])
# show all adjective-noun modifier relationships
for token in parsed:
  if token.dep_ == "amod":
    print('\t'.join(["%d %d"%(token.i, token.head.i), token.text, token.head.text]))
```

```
6 7     first   meal
10 11   wonderful       place
28 30   Best    dish
29 30   deep    dish
35 36   south   side
40 41   great   crust
57 61   fresh   style
58 59   deep    dish
59 61   dish    style
82 83   other   place
85 83   close   place
112 113 Great   service
```

```
spacy_processor = spacy.load("en_core_web_sm")

parsed = spacy_processor(raw_data[1460])
# show all adjective-noun modifier relationships
for token in parsed:
  if token.dep_ == "amod":
    print('\t'.join(["%d %d"%(token.i, token.head.i), token.text, token.head.text]))
```

```
6 7     first   meal
10 11   wonderful       place
28 30   Best    dish
29 30   deep    dish
35 36   south   side
40 41   great   crust
57 61   fresh   style
58 59   deep    dish
59 61   dish    style
82 83   other   place
85 83   close   place
112 113 Great   service
```

mmmmmm. We just finished our first meal at this wonderful place (and by finished I mean until we eat the rest for lunch tomorrow). Best deep dish this side of the south side..... what a great crust, buttery and crunchy even on the bottom. Well worth the wait for a fresh deep dish chicago style! Why, oh why did we wait so long to hear of this seminole heights jewel?! Only other place slightly close to this outside of Chicago is Joey Ds and that is 45 minutes those of us in Tampa dont meed to drive!!! Great service, quaint decor....YUM!!!! We will be back! Take out and cash only... no delivery no CCs. Who cares!!!

Feature Engineering

More Advanced Features

```
y, raw_data = load_data(file)
preprocessed_data = preprocess
X = extract_features(preproces
train_X, train_y, test_X, test_y = split(X, y)
model = train_model(train_X, train_y)
score = evaluate(model, test_X, test_y)
```

Disclaimer: This is pseudocode!

```
def extract_features(dat):
  get_bow_features(dat)
  for i in i..5:
    get_ngrams(dat, i)
  get_dependency_features(dat)
  get_wordnet_features(dat)
```

```
import spacy
!pip install spacy_wordnet
import spacy_wordnet
from spacy_wordnet.wordnet_annotator import WordnetAnnotator
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')

spacy_processor = spacy.load("en_core_web_sm")
spacy_processor.add_pipe("spacy_wordnet", after='tagger')
parsed = spacy_processor(raw_data[1460])

for w in parsed:
  print(w, w._.wordnet.synsets())
```

```
mmmmmm []
. []
We []
just [Synset('merely.r.01'), Synset('precisely.r.01'), Synset('just.r.03'), Synset('just.r.04'), Synset('barely.r
finished [Synset('complete.v.01'), Synset('finish_up.v.02'), Synset('end.v.01'), Synset('finish.v.04'), Synset('e
our []
first [Synset('first.a.01'), Synset('first.s.02'), Synset('inaugural.s.02'), Synset('beginning.s.01'), Synset('fi
meal [Synset('meal.n.01'), Synset('meal.n.02'), Synset('meal.n.03')]
at [Synset('astatine.n.01'), Synset('at.n.02')]
this []
wonderful [Synset('fantastic.s.02')]
place [Synset('topographic_point.n.01'), Synset('place.n.02'), Synset('place.n.03'), Synset('place.n.04'), Synset
```

```
spacy_processor = spacy.load("en_core_web_sm")
spacy_processor.add_pipe("spacy_wordnet", after='tagger')
parsed = spacy_processor(raw_data[1460])

for w in parsed:
  print(w, w._.wordnet.synsets())
  for s in w._.wordnet.synsets():
    print(s, "hypernyms->", s.hypernyms())
```

```
mmmmmm []
. []
We []
just [Synset('merely.r.01'), Synset('precisely.r.01'), Synset('just.r.03'), Synset('just.r.04'), Synset('barely.r
Synset('merely.r.01') hypernyms-> []
Synset('precisely.r.01') hypernyms-> []
Synset('just.r.03') hypernyms-> []
Synset('just.r.04') hypernyms-> []
Synset('barely.r.01') hypernyms-> []
Synset('just.r.06') hypernyms-> []
finished [Synset('complete.v.01'), Synset('finish_up.v.02'), Synset('end.v.01'), Synset('finish.v.04'), Synset('e
Synset('complete.v.01') hypernyms-> [Synset('end.v.02')]
Synset('finish_up.v.02') hypernyms-> [Synset('act.v.01')]
Synset('end.v.01') hypernyms-> []
Synset('finish.v.04') hypernyms-> [Synset('coat.v.01')]
Synset('eat_up.v.01') hypernyms-> [Synset('eat.v.01')]
Synset('finish.v.06') hypernyms-> [Synset('end.v.02')]
```

Feature Engineering

More Advanced Fea

Disclaimer: This is pseudocode!
It doesn't actually run.

```
y, raw_data = loa
preprocessed_data
X = extract_featu
train_X, train_y,
model = train_mod
score = evaluate(
```

```
def extract_features(dat):
  get_bow_features(dat)
  for i in i..5:
    get_ngrams(dat, i)
  get_dependency_features(dat)
  get_wordnet_features(dat)
  get_wikipedia_page_link_features(dat)
  get_twitter_historical_features(data)
  …
```

can be as complex and creative as you want!