Natural Language Processing

AI51701/CSE71001

Lecture 2

08/31/2023

Instructor: Taehwan Kim

Review of the Last Lecture

- ❑ Introduction to the course
- ❑ Examples of Natural Language Processing
- ❑ Word
  - Tokenization

Lecture Plan

- ❑ Tips for Final Projects
- ❑ Word
  - Tokenization
  - Morphology
- ❑ Word representations

Final Project

- Final Project: 40%
  - Proposal (5%): presentation (and written proposal) on Oct. 10 & Oct. 12, feedback will be given.
  - Final presentation (5%): on Dec.5 & Dec. 7
  - Final report (30%): by the end of last week of the semester (Dec. 15)
  - You will choose your project topics.
  - 3 people team
    - For team search, use BlackBoard discussion forum if needed.
  - You can use any language/framework for your project
    - Though we expect most of you to use Python (and recommend PyTorch for deep learning)

Final Project

- What you would do:
  - Defining a research goal
  - Finding data and tools (models)
  - Working out on the problem
  - Evaluating your proposed solution/approach
- Must be human-language related.
  - E.g., applying Transformers to a genetics dataset is **not** allowed unless related to human language pre-training dataset/tasks on downstream performance on the genetics task.

Project Proposal

- ❑ Find a relevant (key) research paper for your topic
- ❑ Write a summary of that research paper and what you took away from it as key ideas that you hope to use
- ❑ Write what you plan to work on and how you can innovate in your final project work
- ❑ Describe as needed
  - A project plan, relevant existing literature, the kind(s) of models you will use/explore (usually deep learning models); the **data** you will use (and how it is obtained), and how you will **evaluate** success
  - Title, team members, and your planned contributions
- ❑(Re-) implementing existing work is not enough.
- ❑3–4 pages (in Latex), presentation (and written proposal via BlackBoard) on Oct. 10 & Oct. 12.

Project Proposal

- Need to think critically about a research paper
  - What were the main novel contributions or points?
  - Is what makes it work something general and reusable or a special case?
  - Are there flaws or neat details in what they did?
  - How does it fit with other papers on similar topics?
  - Does it provoke good questions on further or different things to try?

Choosing Your Topic for Final Project

- Two basic starting points
  - Start with a (domain) problem of interest and try to find good/better ways to address it than are currently known/used
  - Start with a technical method/approach of interest, and work out good ways to extend it, improve it, understand it, or find new ways to apply it

Choosing Your Topic for Final Project

- This is not an exhaustive list, but most projects are one of
  - Find an application/task of interest and explore how to approach/solve it effectively, often with an existing model
  - Implement a complex neural architecture and demonstrate its performance on some data
  - Come up with a new or variant neural network model or approach and explore its empirical success
  - Analysis project. Analyze the behavior of a model: how it represents linguistic knowledge or what kinds of phenomena it can handle or errors that it makes
  - Rare theoretical project: Show some interesting, non-trivial properties of a model type, data, or a data representation

Choosing Your Topic for Final Project

- How to find an interesting place to start?
  - look at online proceedings of major NLP/ML conferences:
    - ACL anthology: https://aclanthology.org/
    - NeurIPS https://papers.nips.cc, ICML https://proceedings.mlr.press/, ICLR https://openreview.net/group?id=ICLR.cc
  - online preprint servers
    - https://arxiv.org
  - look for an interesting problem in the world!
    - Hal Varian: How to Build an Economic Model in Your Spare Time https://people.ischool.berkeley.edu/~hal/Papers/how.pdf

Finding data

- Some people collect their own data for a project
  - You may have a project that uses “unsupervised” data
  - You can annotate a small amount of data
  - You can find a website that effectively provides annotations, such as likes, stars, ratings, responses, etc.
  - But be careful on scoping things so that this doesn’t take most of your time
- **Most people make use of an existing, curated dataset built by previous researchers**
  - You get a fast start and there is obvious prior work and baselines

Useful Resources

- Machine translation
  - http://statmt.org
  - Look in particular at the various WMT shared tasks
- Dependency parsing: Universal Dependencies
  - https://universaldependencies.org
- Huggingface Datasets
  - https://huggingface.co/datasets
- Paperswithcode Datasets
  - https://www.paperswithcode.com/datasets?mod=texts&page=1

From Stanford CS224n course

Useful Resources

- There are now many other datasets available online for all sorts of purposes
  - Look at Kaggle
  - Look at research papers to see what data they use
  - Look at lists of datasets
    https://machinelearningmastery.com/datasets-natural-language-processing/
    https://github.com/niderhoff/nlp-datasets
  - Lots of particular things:
    https://gluebenchmark.com/tasks
    https://nlp.stanford.edu/sentiment/
    https://research.fb.com/downloads/babi/(FacebookbAbI-related)

From Stanford CS224n course

Useful Resources

- Sample projects
  - Stanford 224n course final project reports

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/project.html

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/project.html

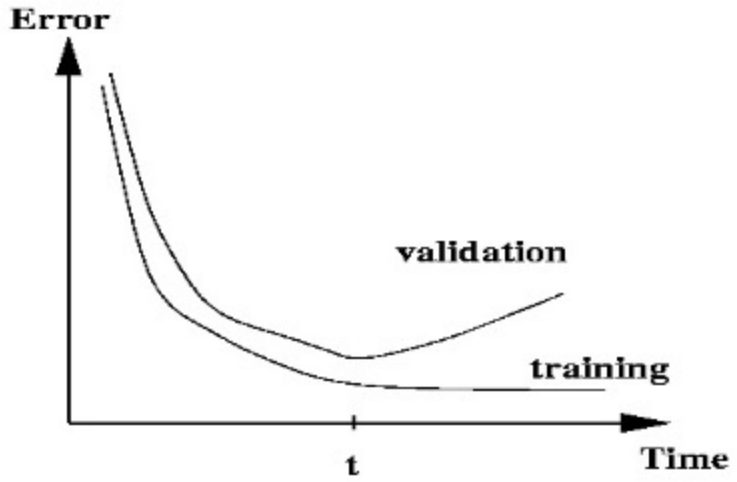
Care with datasets and in model development

- Many publicly available datasets are released with a **train/dev/test** structure
- If there is no dev set or you want a separate tune set, then you create one by splitting the training data
- Having a fixed test set ensures that all systems are assessed against the same gold data.

From Stanford CS224n course

Training models and pots of data

- When training, models overfit to what you are training on
  - The model correctly describes what happened to occur in particular data you trained on, but the patterns are not general enough patterns to be likely to apply to new data
- The way to monitor and avoid problematic overfitting is using **independent** validation and test sets



From Stanford CS224n course

Training models and pots of data

- You build (estimate/train) a model on a training set.
- Often, you then set further hyperparameters on another, independent set of data, the **tuning set**
- You measure progress as you go on a **dev set** (development test set or validation set)
  - If you do that a lot you overfit to the dev set so it can be good to have a second dev set, the **dev2** set
- **Only at the end**, you evaluate and present final numbers on a **test** set

From Stanford CS224n course

Training models and pots of data

- The train, tune, dev, and test sets need to be completely **distinct**
- It is invalid to give results testing on material you have trained on
  - You will get a falsely good performance.
  - We almost always overfit on train

From Stanford CS224n course

Experimental strategy

- Work incrementally!
- Start with a very simple model and get it to work!
  - It's hard to fix a complex but broken model
- Add bells and whistles one-by-one and get the model working with each of them (or abandon them)
- Initially run on a tiny amount of data
  - You will see bugs much more easily on a tiny dataset ... and they train really quickly
  - Something like 4–8 examples is good
  - Often synthetic data is useful for this
  - Make sure you can get 100% on this data

From Stanford CS224n course

Experimental strategy

- There are lots of things that can cause neural nets to not learn at all or to not learn very well
  - Finding and fixing them("debugging and tuning") can often take more time than implementing your model
  - But experience, experimental care, and rules of thumb help!

- Train and run your model on a large dataset
  - It should still score close to 100% on the training data after optimization.
    - Otherwise, you probably want to consider a more powerful model!
    - Overfitting to training data is **not** something to fear when doing deep learning

- You now want good generalization performance:
  - Regularize your model until it doesn't overfit on dev data
  - Strategies like L2 regularization can be useful
  - But normally generous **dropout** is the secret to success

From Stanford CS224n course

Details matter!

- Look at your **data**, collect summary statistics
- Look at your model's outputs, do error **analysis**
- Tuning hyperparameters, learning rates, getting initialization right, etc. is **often** important to the successes of Neural Networks

From Stanford CS224n course

Good luck with your projects!

Word

Tokenization (review)

- Tokenization: convert a character stream into words by adding spaces
  - For certain languages, highly nontrivial
  - E.g., Chinese word segmentation is a widely- studied NLP task
  - for other languages (English), tokenization is easier but is still not always obvious

Another option for text tokenization

- Instead of
  - white-space segmentation
  - single-character segmentation

- **Use the data** to tell us how to tokenize

- **Subword tokenization** (because tokens can be parts of words as well as whole words)

Subword tokenization

- Three common algorithms: ◦
  - Byte-Pair Encoding (BPE) (Sennrich et al., 2016)
  - Unigram language modeling tokenization (Kudo, 2018)
  - WordPiece (Schuster and Nakajima, 2012)
- All have 2 parts:
  - A token learner that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - A token segmenter that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

- Let vocabulary be the set of all individual characters

  = {A, B, C, D,…, a, b, c, d….}

- Repeat:
  - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- Until k merges have been done.

BPE token learner

Original (very fascinating🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

vocabulary

_, d, e, i, l, n, o, r, s, t, w

BPE token learner

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Merge e r to er

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

BPE

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2	l o w e s t _	
6	n e w er _	
3	w i d er _	
2	n e w _	

Merge er _ to er_

corpus		vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

BPE

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_
2 l o w e s t _	
6 n e w er_	
3 w i d er_	
2 n e w _	

Merge n e to ne

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne
2 l o w e s t _	
6 ne w er_	
3 w i d er_	
2 ne w _	

BPE

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE token segmenter algorithm

- On the test data, run each merge learned from the training data:
  - Greedily
  - In the order we learned them
  - (test frequencies don't play a role)
- So: merge every e r to er, then merge er _ to er_, etc.
- Result:
  - Test set "n e w e r _" would be tokenized as a full word
  - Test set "l o w e r _" would be two tokens: "low er_

Properties of BPE tokens

- Usually include frequent words
- And frequent subwords
  - Which are often morphemes like -est or –er

- A **morpheme** is the smallest meaning-bearing unit of a language
  - unlikeliest has 3 morphemes un-, likely, and -est

Word structure and subword models

- Let's take a look at the assumptions we've made about a language's vocabulary.
- We assume a fixed vocab of tens of thousands of words, built from the training set.
- All *novel* words seen at test time are mapped to a single UNK.

	word		vocab mapping
Common words	hat	→	hat
	learn	→	learn
Variations	taaaaasty	→	UNK
misspellings	laern	→	UNK
novel items	Transformerify	→	UNK

From Stanford CS224n course

Word structure and subword models

- Finite vocabulary assumptions make even **less** sense in many languages.
- Many languages exhibit complex morphology, or word structure.
  - The effect is more word types, each occurring fewer times.

Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++)

Here's a small fraction of the conjugations for *ambia* – to tell.

Conjugation of *-ambia* [less ▲]

Non-finite forms		
Form	Positive	Negative
Infinitive	kuambia	kutoambia
Simple finite forms		
Positive form	Singular	Plural
Imperative	ambia	ambieni
Habitual	huambia	

Complex finite forms

Polarity	Persons				Persons / Classes		Classes											
	1st		2nd		3rd / M-wa		M-mi	Ma			Ki-vi		N		U	Ku	Pa	Mu
	Sg.	Pl.	Sg.	Pl.	Sg. / 1	Pl. / 2	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18
Past [less ▲]																		
Positive	niliambia naliambia	tuliambia twaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia
Present [less ▲]																		
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii
Future [less ▲]																		
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia
Subjunctive [less ▲]																		
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie
Present Conditional [less ▲]																		
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia
Negative	nisingeambia singeambia	tusingeambia hatungeambia	usingeambia hungeambia	msingeambia hamngeambia	asingeambia hangeambia	wasingeambia hawangeambia	usingeambia haungeambia	isingeambia haingeambia	lisingeambia halingeambia	yasingeambia hayangeambia	kisingeambia hakingeambia	visingeambia havingeambia	isingeambia haingeambia	zisingeambia hazingeambia	usingeambia haungeambia	kusingeambia hakungeambia	pasingeambia hapangeambia	musingeambia hamungeambia
Past Conditional [less ▲]																		
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia
Negative	nisingaliambia singaliambia	tusingaliambia hatungaliambia	usingaliambia hungaliambia	msingaliambia hamngaliambia	asingaliambia hangaliambia	wasingaliambia hawangaliambia	usingaliambia haungaliambia	isingaliambia haingaliambia	lisingaliambia halingaliambia	yasingaliambia hayangaliambia	kisingaliambia hakingaliambia	visingaliambia havingaliambia	isingaliambia haingaliambia	zisingaliambia hazingaliambia	usingaliambia haungaliambia	kusingaliambia hakungaliambia	pasingaliambia hapangaliambia	musingaliambia hamungaliambia
Conditional Contrary to Fact [less ▲]																		
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia
Gnomic [less ▲]																		
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia
Perfect [less ▲]																		

[Wiktionary]

From Stanford CS224n course

The byte-pair encoding algorithm

- Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)
  - The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens).
  - At training and testing time, each word is split into a sequence of known subwords.
- **Byte-pair encoding** is a simple, effective strategy for defining a subword vocabulary.
- Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

[Sennrich et al., 2016, Wu et al., 2016]

From Stanford CS224n course

Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- In the worst case, words are split into as many subwords as they have characters.

	word		vocab mapping
Common words	hat	→	hat
	learn	→	learn
Variations	taaaaasty	→	taa## aaa## sty
misspellings	laern	→	la## ern
novel items	Transformerify	→	Transformer## ify

From Stanford CS224n course

Morphology

- Words are not atoms
  - They have internal structure
  - They are composed (to a first approximation) of **morphemes**

- **Morphemes** are minimal meaningful components of words.
  - **Stems** (roots) : The central morphemes in words, which carry the main meaning
  - **Affixes**: bits and pieces that adhere to stems
    - Prefixes: **pre**-nuptual, **ir**-regular
    - Suffixes: determin-**ize**, iterat-**or**
    - Infixes: cup-**s**-ful
    - Circumfixes: **ge**-sammel-**t**

Morphology

- Type/Token Ratio across Languages
  - high type/token ratio -> rich morphology
  - low type/token ratio -> poor morphology

Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols:
  - Such symbols for words can be represented by one-hot vectors:
    - E.g., motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0], hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
  - Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with words as discrete symbols

- Example: in web search, if a user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”
  - But these two vectors are orthogonal
    - motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0], hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
- There is no natural notion of similarity for one-hot vectors!
- Instead, learn to encode similarity in the vectors themselves

Intuitions of Distributional Models

- Suppose I gave you the following corpus:
  - A bodle of ***tesgüino*** is on the table
  - Everybody likes ***tesgüino***
  - ***Tesgüino*** makes you drunk
  - We make ***tesgüino*** out of corn.

- What is ***tesgüino****?*
- From context, we can guess ***tesgüino*** is an alcoholic beverage like beer
- Intuition: two words are similar if they have similar word contexts

Many ways to get word vectors

- Some based on counting, some based on prediction/learning
- Some sparse, some dense
- Some have interpretable dimensions, some don't

- Shared ideas:
  - model meaning of a word by "embedding" it in a vector space
  - these word vectors are also called "**embeddings**"

- Contrast: in traditional NLP, word meaning is represented by a vocabulary index ("word #545")

Distributional Word Vectors

- We'll start with the simplest way to create word vectors:
- Count occurrences of context words
  - so, vector for *pineapple* has counts of words in the context of *pineapple* in a dataset
  - one entry in vector for each unique context word
  - stack these vectors for all words in a vocabulary *V* to produce a count matrix *C*
  - *C* is called the **word-context matrix** (or **word- word co-occurrence matrix**)

Counting Context Words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
r enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	...
pineapple	0	0	0	1	0	1	...
digital	0	2	1	0	1	0	...
information	0	1	6	0	4	0	...
...							

Word-Context Matrix

- We showed 4x6, but actual matrix is $|V| \times |V|$
  - Very large, but very **sparse** (mostly zeroes)
  - Lots of efficient algorithms for sparse vectors and matrices

- Context Window Size
  - Size of context window affects word vectors

Measuring similarity

- Given 2 word vectors, how should we measure their similarity?
- Most measure of vector similarity are based on **dot product** (or **inner product**):

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^\top \mathbf{v} = \sum_i u_i v_i$$

  - High when vectors have large values in same dimensions

- Notation

$\mathbf{u} =$ a vector

$u_i =$ entry i in the vector

$\mathbf{u}^\top \mathbf{v} =$ dot (inner) product

Problem with dot product?

$$\mathbf{u}^\top \mathbf{v} = \sum_i u_i v_i$$

- Dot product is larger if vector is longer
- Vector length:

$$||\mathbf{u}|| = \sqrt{\sum_i u_i^2}$$

- Frequent words → larger counts → larger dot products
- This is bad: we don't want a similarity metric to be overly sensitive to word frequency

Solution: cosine similarity

- Divide dot product by lengths of the vectors

$$\frac{\mathbf{u}^\top \mathbf{v}}{||\mathbf{u}||\,||\mathbf{v}||}$$

- Turns out to be the cosine of the angle between them!

Problems with raw counts

- Raw word counts are not a great measure of association between words
  - very skewed: *the* and *of* are frequent, but not the most discriminative

Top co-occurrence counts with ``cooked''

123	,
92	and
79	the
71	.
68	<s>
66	</s>
53	in
39	a
38	is
35	of
30	with
28	are
25	to
23	or
23	it
20	(
19	be
15	)
14	"

13	as
12	for
12	food
11	which
11	that
11	meat
11	can
11	by
10	when
9	rice
9	raw
9	beef
7	they
7	their
7	on
7	not
7	from
6	leaves
6	has

Top co-occurrence counts with ``cooked''

123	,
92	and
79	the
71	.
68	<s>
66	</s>
53	in
39	a
38	is
35	of
30	with
28	are
25	to
23	or
23	it
20	(
19	be
15	)
14	"

13	as
12	for
12	**food**
11	which
11	that
11	**meat**
11	can
11	by
10	when
9	**rice**
9	**raw**
9	**beef**
7	they
7	their
7	on
7	not
7	from
6	**leaves**
6	has

Problems with raw counts

- Raw word counts are not a great measure of association between words
  - very skewed: *the* and *of* are frequent, but not the most discriminative

- Rather have a measure that asks whether a context word is informative about the center word
  - **pointwise mutual information** (PMI)

Pointwise Mutual Information (PMI)

- PMI has been used for finding **collocations** and **associations** between words

$$\mathrm{pmi}(x;y) = \log_2 \frac{p(x,y)}{p(x)p(y)}$$

- Here, *x* is the center word and *y* is the word in the context window
- Each of these probabilities can be estimated from the counts collected from the corpus
- Replace raw counts with PMI scores

Context words of “cooked” with highest PMIs

9.30533	beef
8.88418	shrimp
8.63397	potatoes
8.61946	ate
8.56584	dishes
8.50945	eaten
8.4931	beans
8.33137	texture
8.29489	vegetables
8.25088	soda
8.20831	meat
8.15708	sauce
8.08345	consuming
7.9532	cuisine
7.94043	raw
7.78435	curry
7.7563	juice
7.74444	vegetable
7.66406	chili
7.56264	rice
7.56167	soup
7.45315	flour
7.43874	steamed
7.43715	crushed
7.41193	meals
7.39793	digest
7.39175	rockies
7.34773	ramsay
7.33211	honey
7.32253	toxicity
7.29057	cared
7.28626	tomatoes
7.27912	boiling
7.27769	dal
7.27485	citrus
7.25649	doncaster

Positive Pointwise Mutual Information (PPMI)

- PMI ranges from –infinity to +infinity
- But negative values are problematic:
  - things are co-occurring **less than** we expect by chance
  - unreliable without enormous corpora

- So we sometimes replace negative PMI values by 0, calling it positive PMI (PPMI)

Alternative to PPMI

- TF-IDF
- Product of two factors:
  - term frequency (TF; Luhn, 1957): count of word (or possibly log of count)
  - **inverse document frequency** (IDF; Sparck Jones, 1972)
    - *N:* total number of documents
    - df(*x*): # of documents with word *x*

$$\mathrm{idf}(x) = \log \frac{N}{\mathrm{df}(x)}$$

Sparse vs. dense vectors

- So far, our vectors are
  - **long** (length = 25,000)
  - **sparse** (mostly zero)

- Why might we want to reduce vector dimensionality?

Why reduce dimensionality?

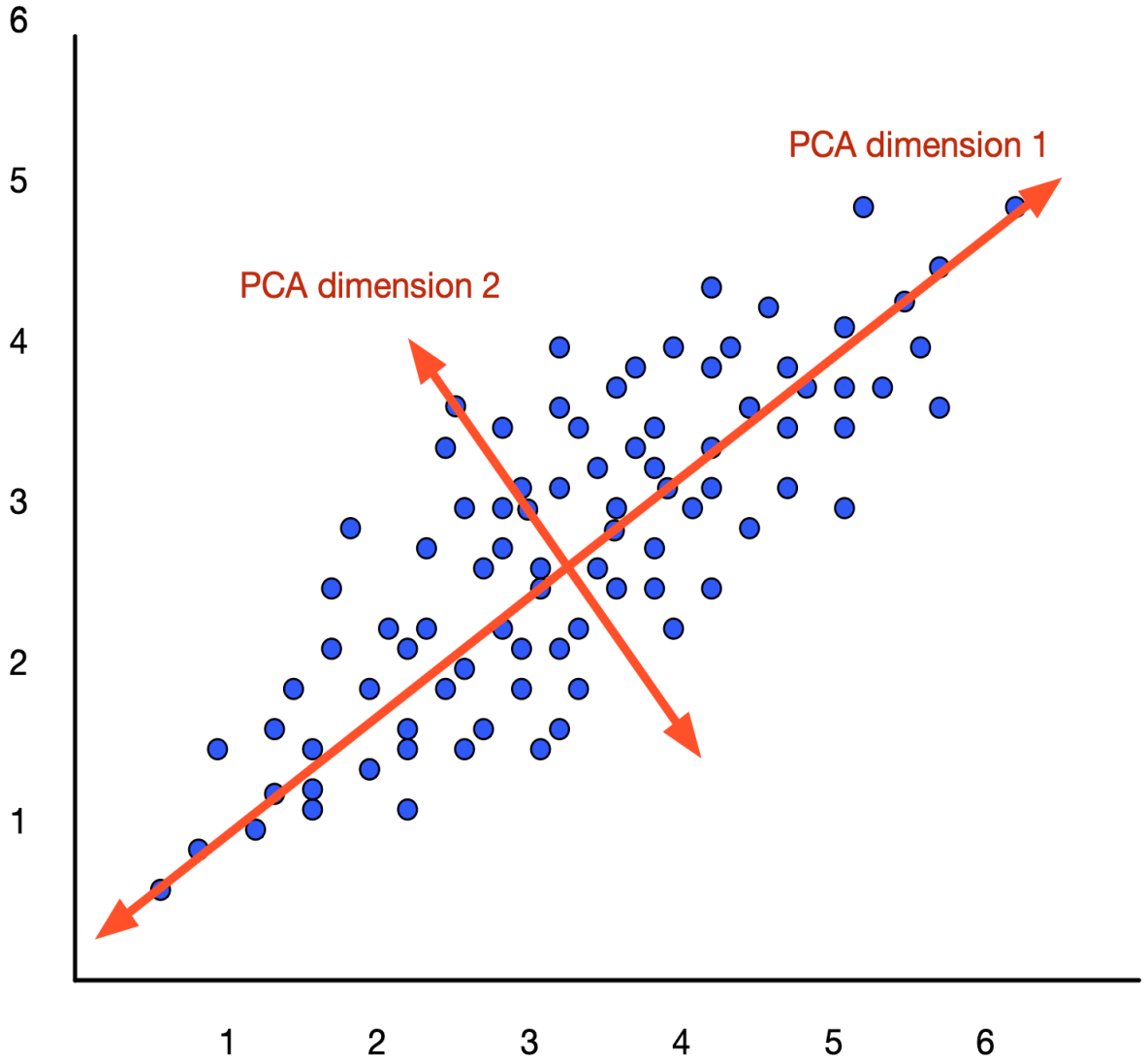
- Short vectors may be easier to use as features (fewer weights to tune)
- Reducing dimensionality may better handle variability in natural language due to synonymy:
  - car and *automobile* are synonyms, but are distinct dimensions
  - fails to capture similarity between a word with *car* as a neighbor and one with *automobile* as a neighbor

Dimensionality Reduction: Intuition

- Approximate an *N*-dimensional dataset using fewer dimensions:
  - rotate axes into a new space
  - in which first dimension captures most variance in original dataset

- Many such (related) methods:
  - principal component analysis (PCA)
  - factor analysis
  - singular value decomposition (SVD)

Dimensionality Reduction



SVD embeddings vs. sparse vectors

- Dense SVD embeddings sometimes work better than sparse PMI vectors at tasks (like word similarity)
  - denoising: low-order dimensions may represent unimportant information
  - truncation may help the models generalize better to unseen data
  - smaller number of dimensions may make it easier for classifiers to effectively assign weights to dimensions for the task
  - dense models may do better at capturing higher order co-occurrence

How should we evaluate word vectors?

- WordSim353 (Finkelstein et al., 2002)

word pair		similarity
journey	voyage	
king	queen	
computer	software	
law	lawyer	
forest	graveyard	
rooster	voyage	

How should we evaluate word vectors?

- WordSim353 (Finkelstein et al., 2002)

Instructions:

Assign a numerical similarity score between 0 and 10
(0 = words are totally unrelated,
10 = words are VERY closely related).

When estimating similarity of antonyms, consider them "similar" (i.e., belonging to the same domain or representing features of the same concept), **rather than "dissimilar"**.

forest	graveyard	
rooster	voyage	

How should we evaluate word vectors?

- WordSim353 (Finkelstein et al., 2002)

word pair		similarity
journey	voyage	9.3
king	queen	8.6
computer	software	8.5
law	lawyer	8.4
forest	graveyard	1.9
rooster	voyage	0.6

How should we evaluate word vectors?

- SimLex-999 (Hill et al., 2014)

word pair		similarity
insane	crazy	9.6
attorney	lawyer	9.4
author	creator	8.0
diet	apple	1.2
new	ancient	0.2

measures **paraphrastic** similarity:

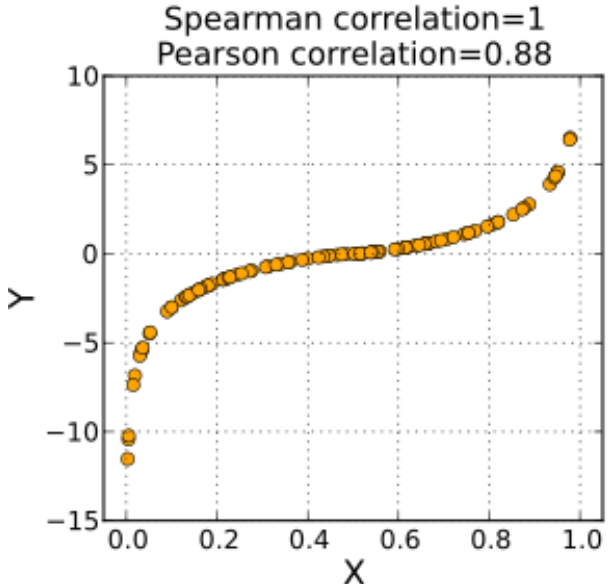
two words are “similar” if they have similar meanings

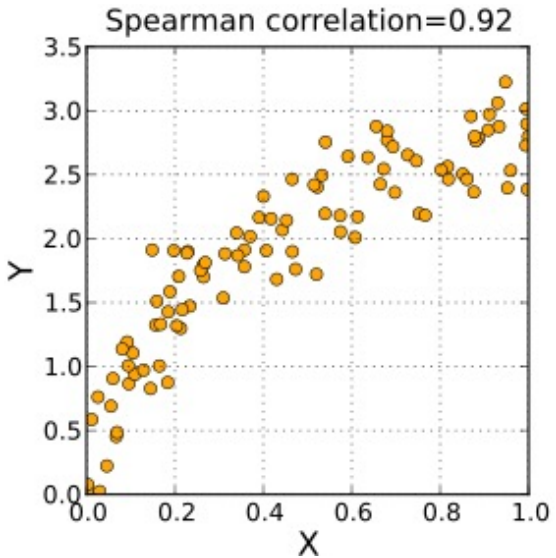
How should we evaluate word vectors?

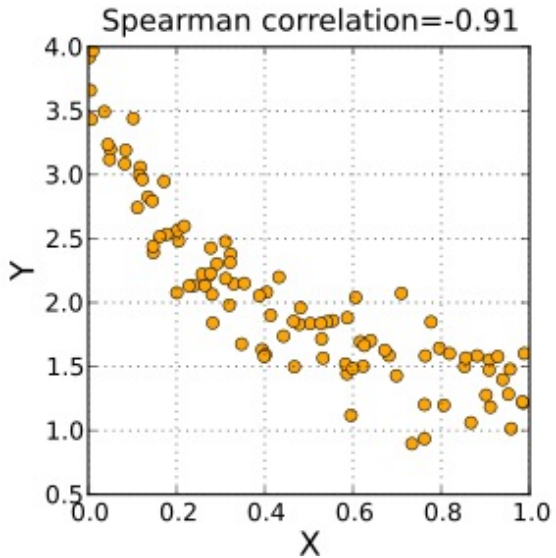
- There are many word similarity datasets
- Some focus on topical **relatedness**, others focus on similarity in **meaning**
- In assignment 1, you will evaluate your word vectors using MEN (relatedness) and SimLex-999 (meaning)

Evaluation Metrics for Word Similarity

- Spearman rank correlation coefficient
- Measures correlation between two variables:
  - variable 1: human-annotated similarities for word pairs
  - variable 2: cosine similarities computed with your word vectors for the same word pairs





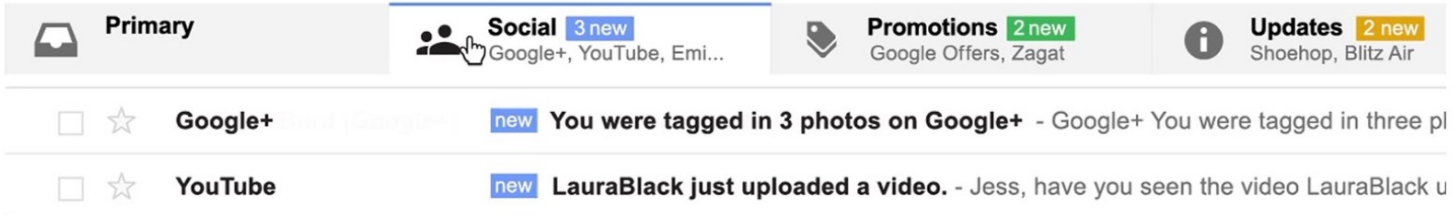


Wikipedia

Text Classification

Text Classification

- ❑ Simplest user-facing NLP application
- ❑ Email (spam, priority, categories):



- ❑ Sentiment:

- ❑ Topic classification
- ❑ Others?

Text Classification

- **Datasets**
- Classification
  - Modeling
  - Inference
  - Learning

NLP Datasets

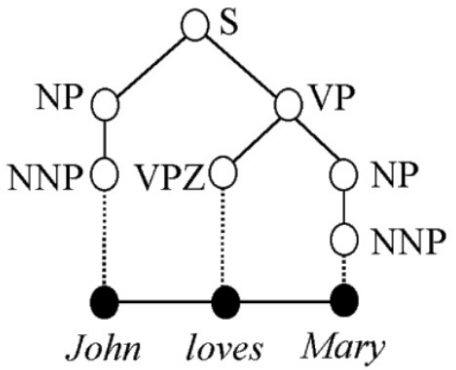
- NLP datasets include inputs (usually text) and outputs (usually some sort of **annotation**)

Annotation

- Supervised machine learning needs labeled datasets, where labels are called **ground truth**
- In NLP, labels are annotations provided by humans
- There is always some disagreement among annotators, even for simple tasks
- These annotations are called a **gold standard**, not ground truth

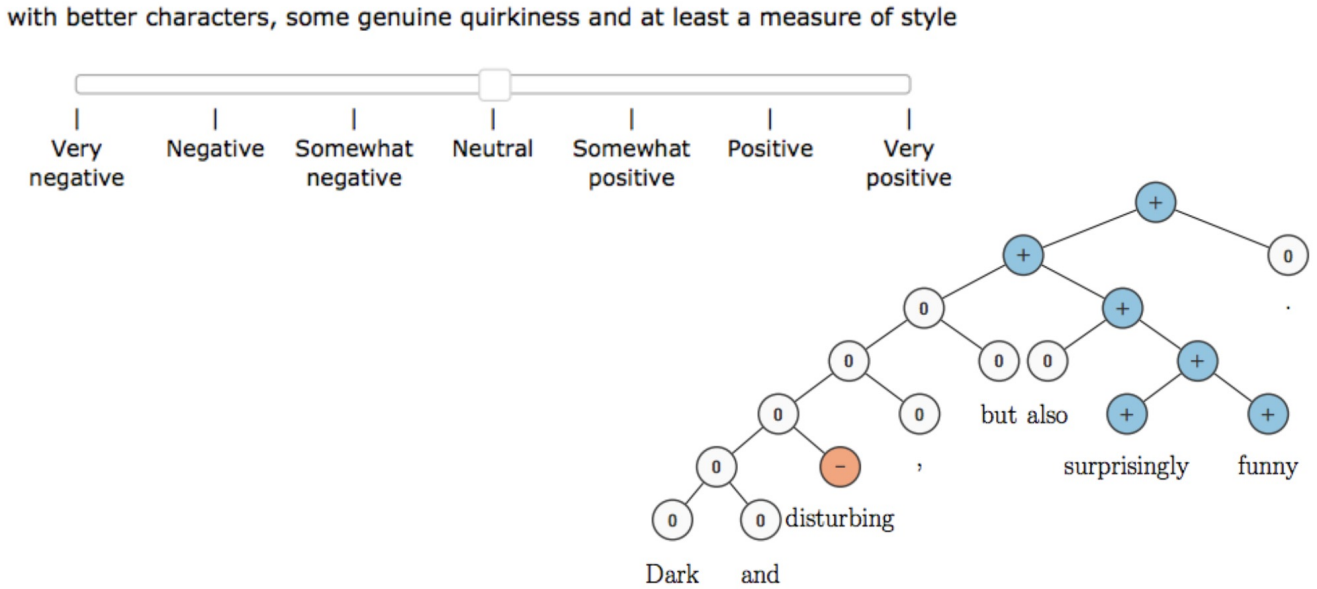
How are NLP datasets developed?

- Paid, trained human annotators
  - traditional approach
  - researchers write annotation guidelines, recruit & pay annotators (often linguists)
  - more consistent annotations, but costly to scale
  - e.g., Penn Treebank (1993)
    - 1 million words, mostly Wall Street Journal, annotated with part-of-speech tags and syntactic parse trees



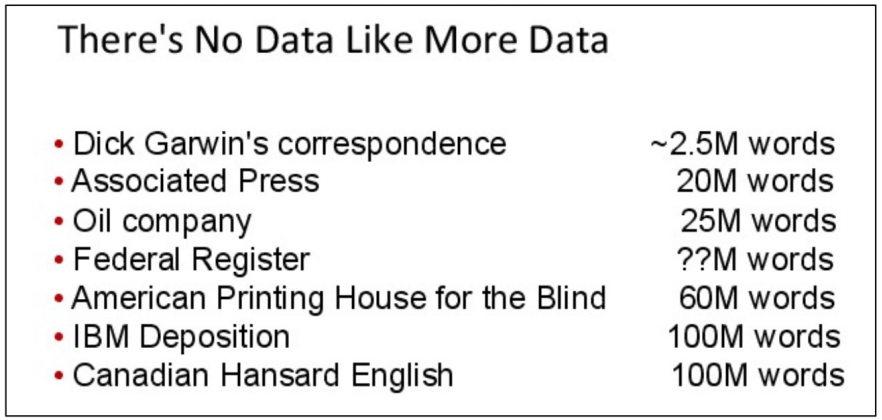
Crowdsourcing

- Crowdsourcing (e.g., Amazon Mechanical Turk)
  - more recent trend
  - can't really train annotators, but easier to get multiple annotations for each input (which can then be averaged)
  - e.g., Stanford Sentiment Treebank



Naturally-occurring annotation

- Long history: used by IBM for speech recognition



credit: Brown & Mercer, 20 Years of Bitext Workshop, 2013

- How might you find naturally-occurring data for:
  - conversational agents
  - summarization
  - coreference resolution

Annotator Agreement

- given annotations from two annotators, how should we measure inter-annotator agreement?
  - percent agreement?
  - Cohen's Kappa (Cohen, 1960) accounts for agreement by chance
  - generalizations exist for more than two annotators (Fleiss, 1971)

Text Classification Data

- There are many annotated datasets
  - Stanford Sentiment Treebank: fine-grained sentiment analysis of movie reviews
  - subjectivity/objectivity sentence classification
  - binary sentiment analysis of customer reviews

Subjectivity/objectivity classification:

the hulk is an anger fueled monster with incredible strength and resistance to damage .	objective
in trying to be daring and original , it comes off as only occasionally satirical and never fresh .	subjective
solondz may well be the only one laughing at his own joke	subjective
obstacles pop up left and right , as the adventure gets wilder and wilder .	objective

- How was this dataset generated?
  - IMDB plot summaries: objective
  - Rotten Tomatoes snippets: subjective

customer review sentiment classification:

it works with a minimum of fuss .	positive
i 've had this thing just over a month and the headphone jack has already come loose .	negative
size - bigger than the ipod	negative
you can manage your profile , change the contrast of backlight , make different type of display , either list or tabbed .	positive
i replaced it with a router raizer and it works much better .	negative

Question classification:

Who invented baseball ?	human
CNN is an acronym for what ?	abbreviation
Which Latin American country is the largest ?	location
How many small businesses are there in the U.S .	number
What would you add to the clay mixture to produce bone china ?	entity
What is the root of all evil ?	description

Text Classification

- Datasets
- **Classification**
  - Modeling
  - Inference
  - Learning

What is a classifier?

- A function from inputs ***x*** to classification labels *y*
- One simple type of classifier:
  - for any input ***x***, assign a score to each label *y,* parameterized by parameters ***w***:

$$\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

- Notation

$\mathbf{u} =$ a vector

$u_i =$ entry i in the vector

$\boldsymbol{x} =$ a structured object

$x_i =$ entry i in the structured object

What is a classifier?

- A function from inputs ***x*** to classification labels *y*
- One simple type of classifier:
  - for any input ***x***, assign a score to each label *y,* parameterized by parameters ***w***:

$$\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

  - classify by choosing highest-scoring label:

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \quad \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Modeling, Inference, Learning

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \;\; \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

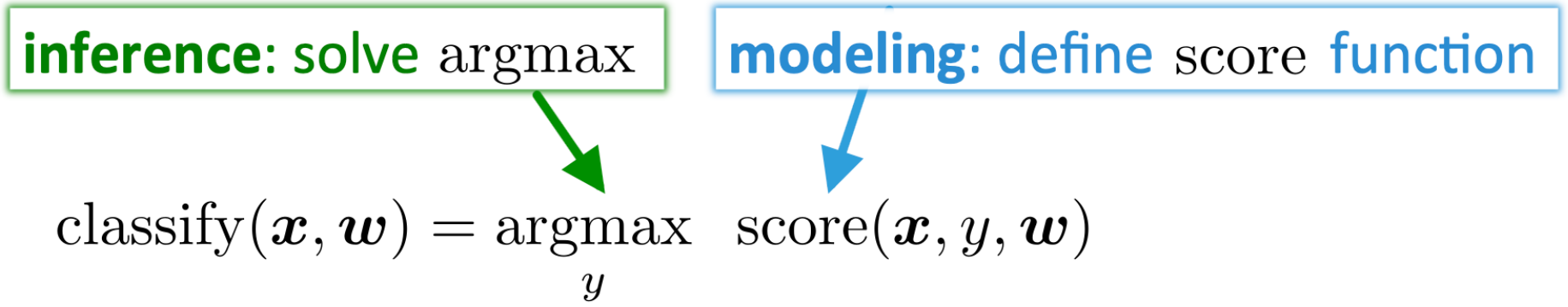
Modeling, Inference, Learning

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \quad \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

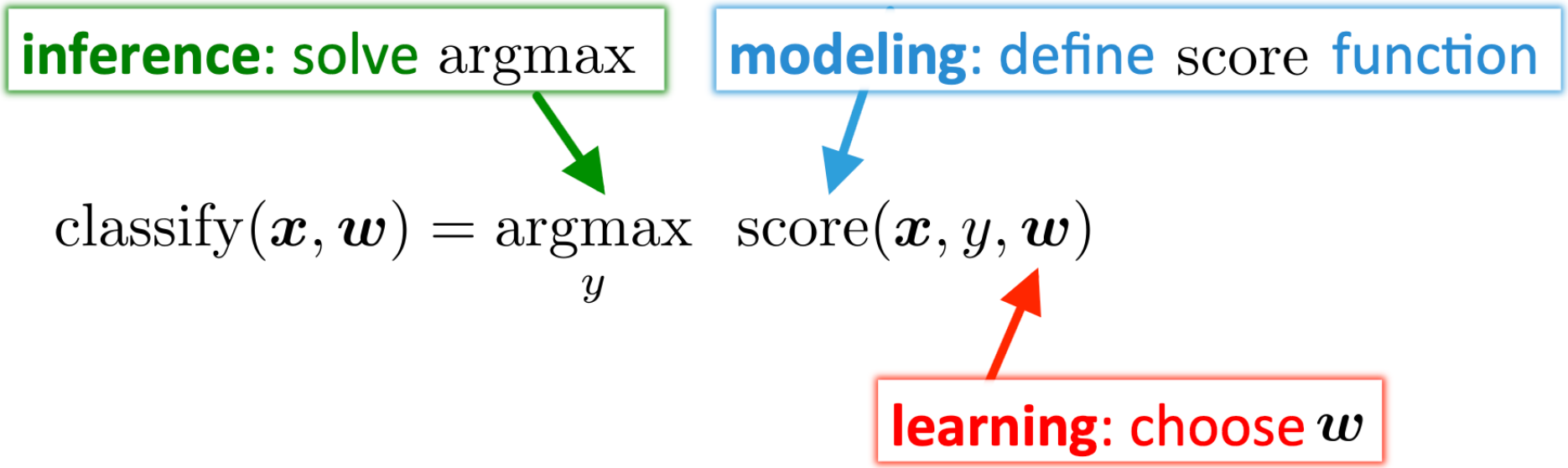
- **Modeling**: How do we assign a score to an (x,y) pair using parameters w?

Modeling, Inference, Learning



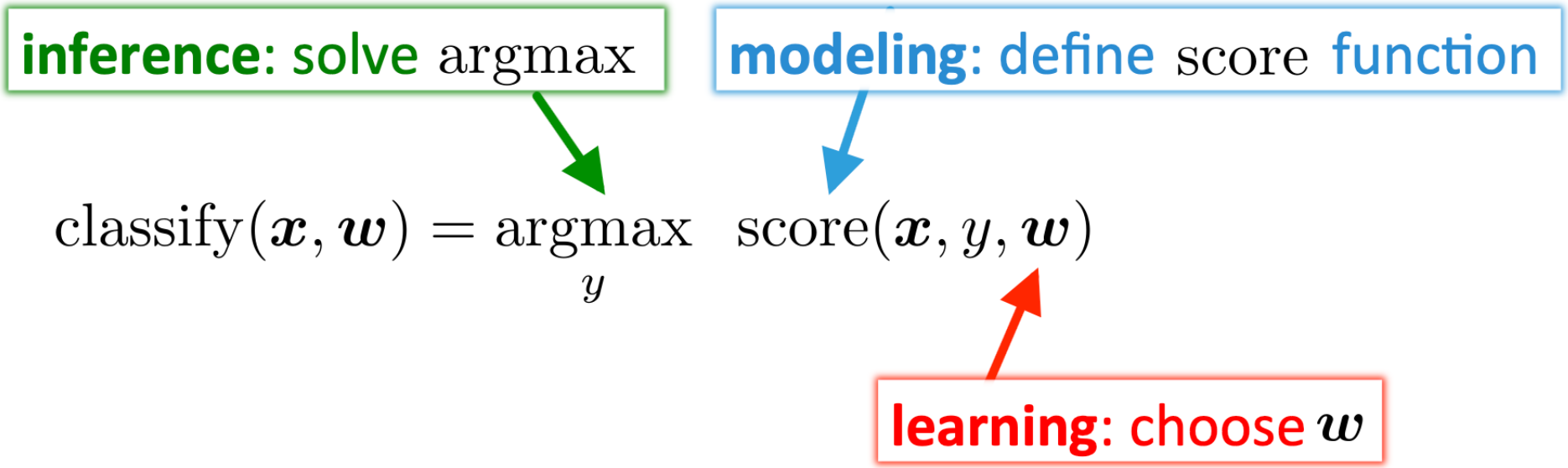
- **Inference**: How do we efficiently search over the space of all labels?

Modeling, Inference, Learning



- **Learning**: How do we choose the weights ***w***?

Modeling, Inference, Learning



- We will use this formulation throughout
  - even when output space is exponentially large or unbounded (e.g., machine translation)

Binary Sentiment Classification

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y}{\text{argmax}} \quad \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Binary Sentiment Classification

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y \in \{\text{positive,negative}\}}{\text{argmax}} \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

a sequence of words
(a sentence or document)

sentiment label

Binary Sentiment Classification

modeling: define score function

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y \in \{\text{positive,negative}\}}{\text{argmax}} \; \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$

Linear Models

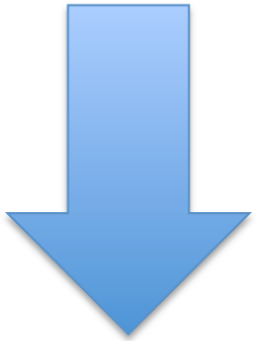
- Parameters are arranged in a vector **w**
- score function is linear in **w**:

$$\text{score}(\boldsymbol{x}, y, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y) = \sum_i w_i f_i(\boldsymbol{x}, y)$$

- **f**: vector of feature functions

Linear Models for Binary Sentiment Classification

$$\text{classify}(\boldsymbol{x}, \boldsymbol{w}) = \underset{y \in \{\text{positive,negative}\}}{\text{argmax}} \quad \text{score}(\boldsymbol{x}, y, \boldsymbol{w})$$



$$\text{classify}(\boldsymbol{x}, \mathbf{w}) = \underset{y \in \{\text{positive,negative}\}}{\text{argmax}} \quad \mathbf{w}^\top \mathbf{f}(\boldsymbol{x}, y)$$

- How do we define **f** ?