Natural Language Processing

AI51701/CSE71001

Lecture 3

09/05/2023

Instructor: Taehwan Kim

Questions from Students

- Topic
  - Multimodal is also good as long as human language is involved

- When to form your project team
  - You have time but now is better

Review of the Last Lecture

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

Lecture Plan

- ❑ Word
  - Word representations

- ❑ Text Classification

Word

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

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

- 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*

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

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?

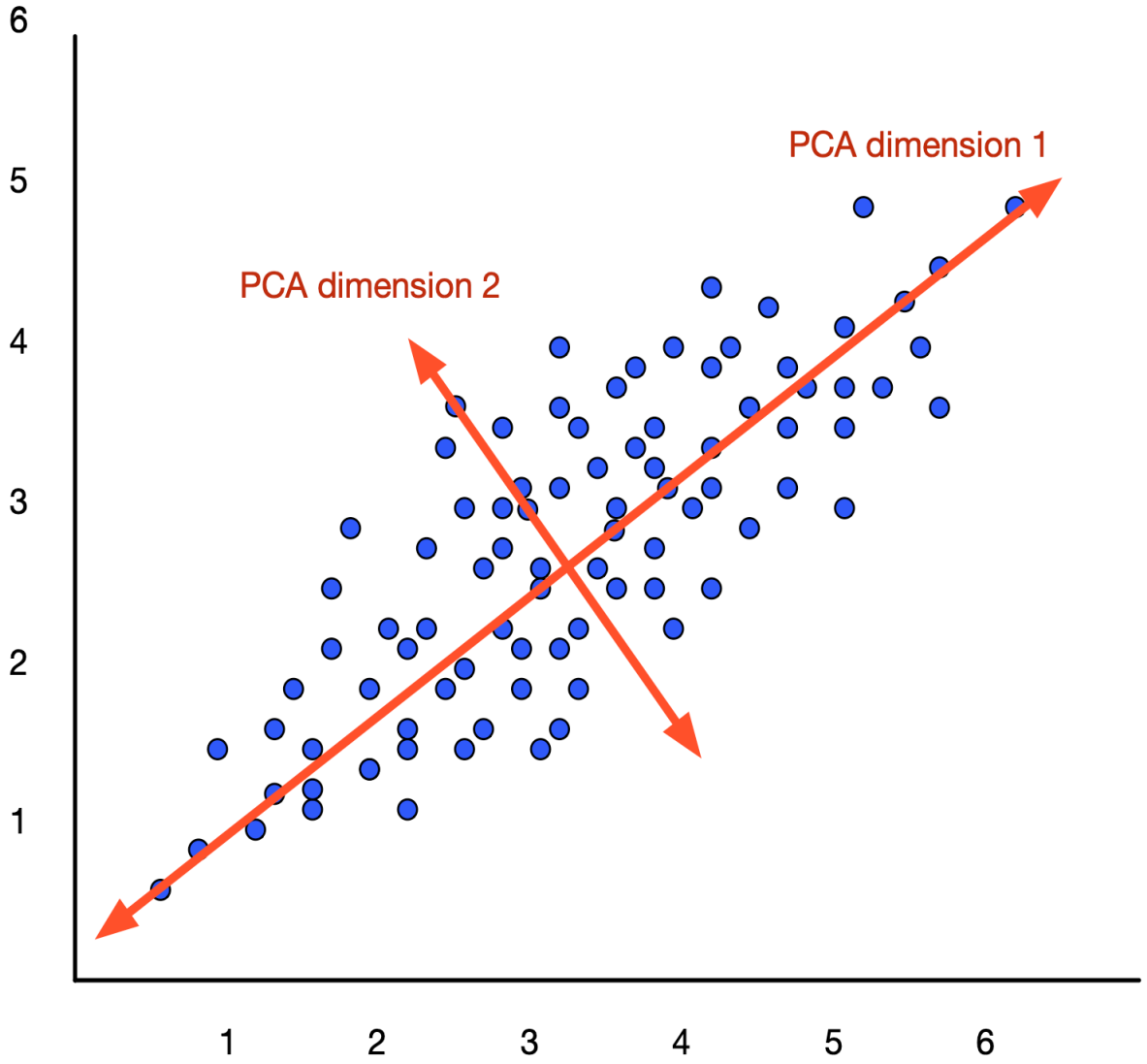
- Short vectors may be easier to use as features (fewer weights to tune)
- Reducing dimensionality may better handle variability in natural language due to synonymy:
  - car and *automobile* are synonyms, but are distinct dimensions
  - fails to capture similarity between a word with *car* as a neighbor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

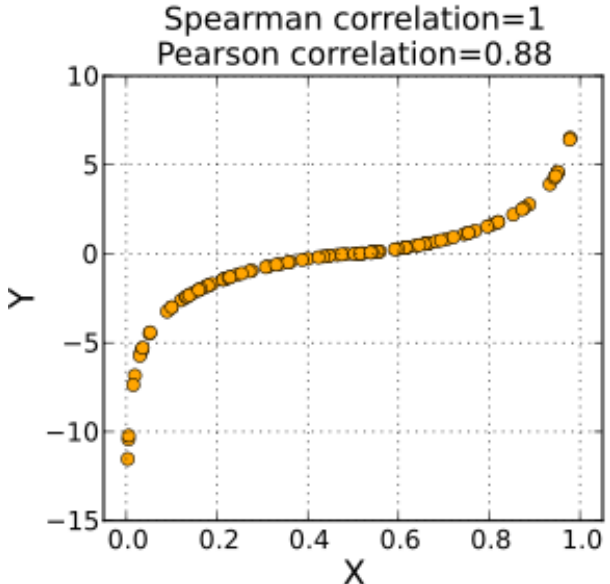
two words are “similar” if they have similar meanings

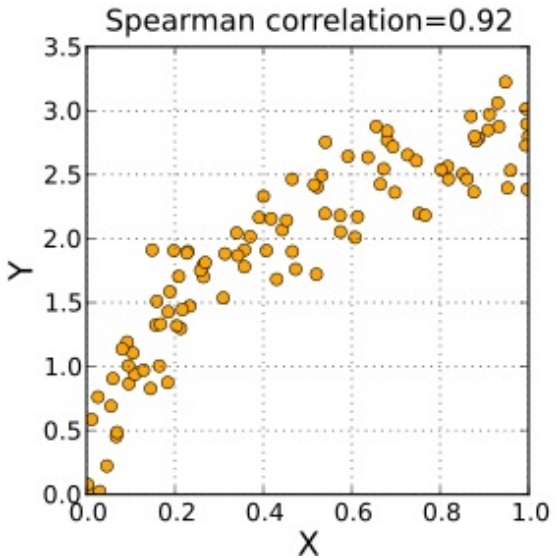
How should we evaluate word vectors?

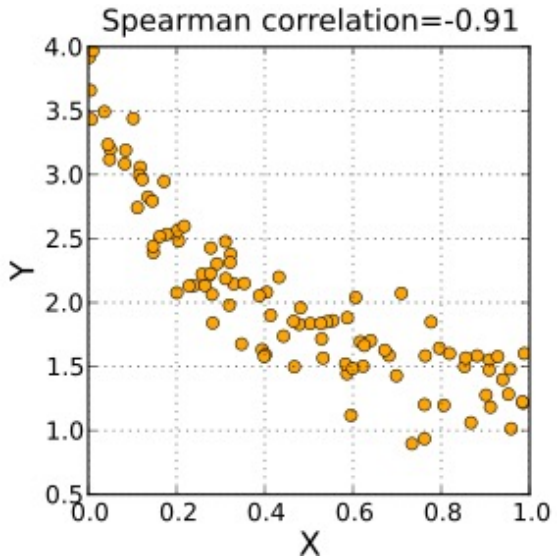
- There are many word similarity datasets
- Some focus on topical **relatedness**, others focus on similarity in **meaning**

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





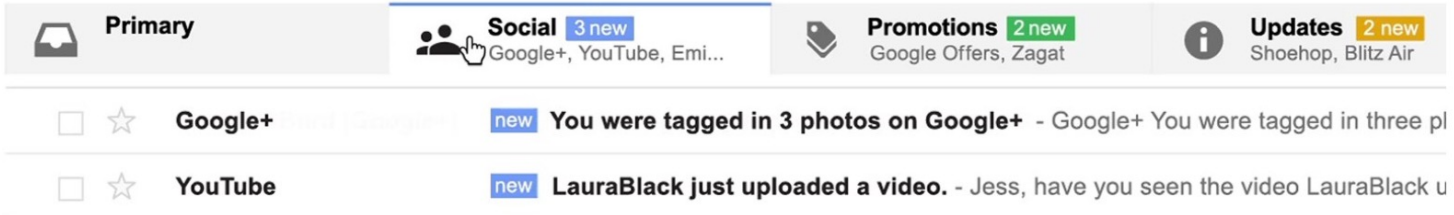


From Wikipedia

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

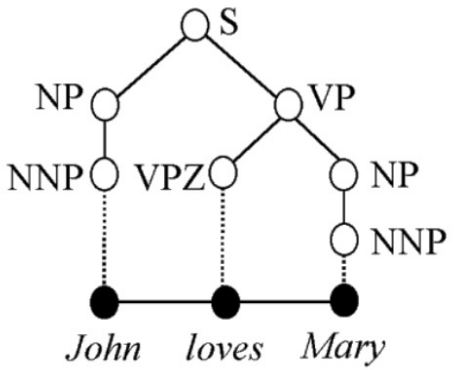
- NLP datasets include inputs (usually text) and outputs (usually some sort of **annotation**)

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

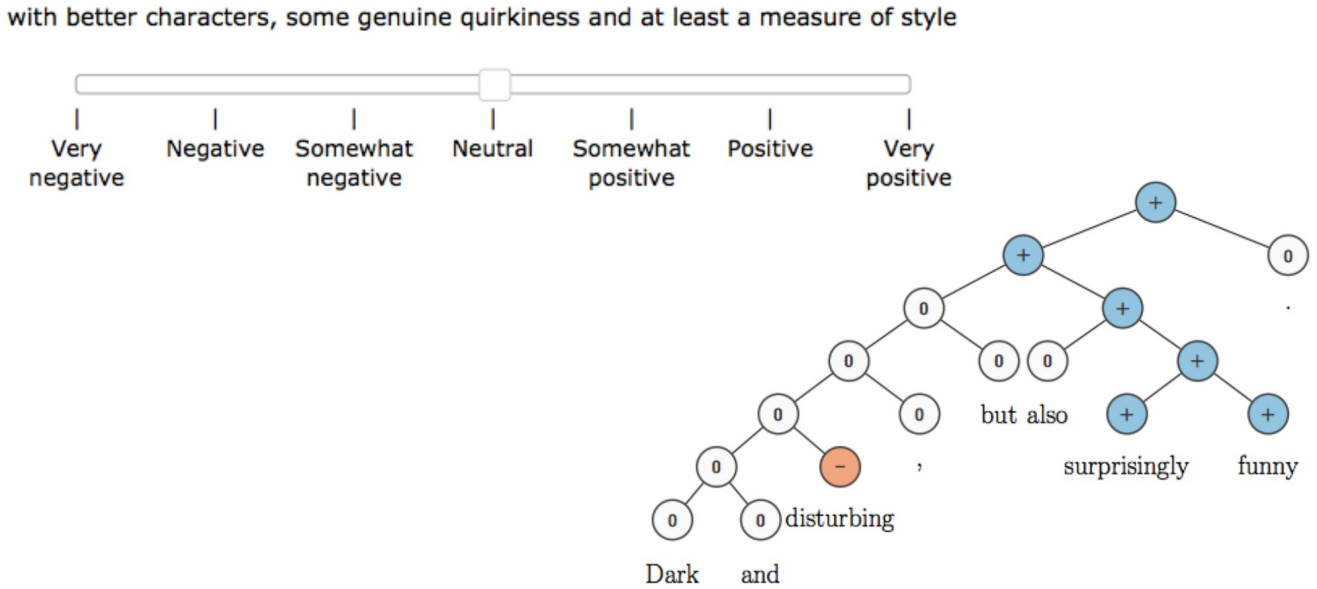
How are NLP datasets developed?

- 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



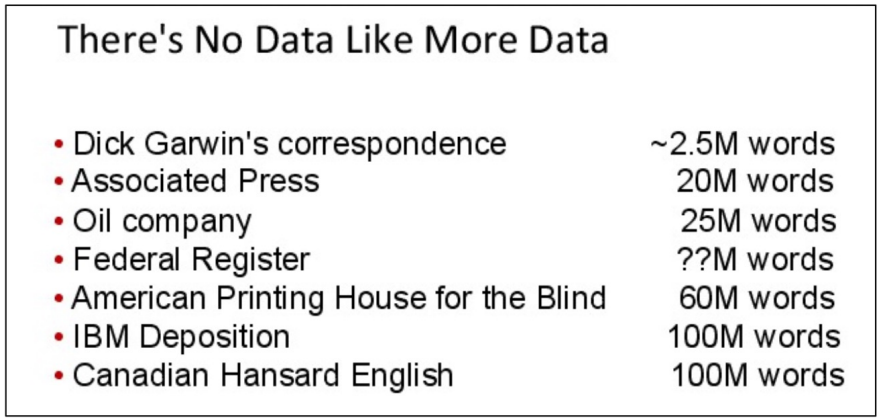
Crowdsourcing

- 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

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

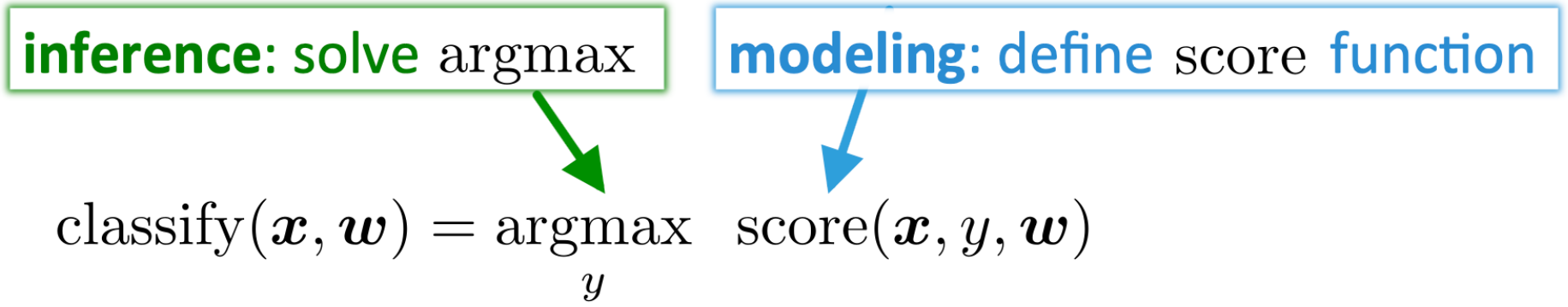
Modeling, Inference, Learning

modeling: define score function

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

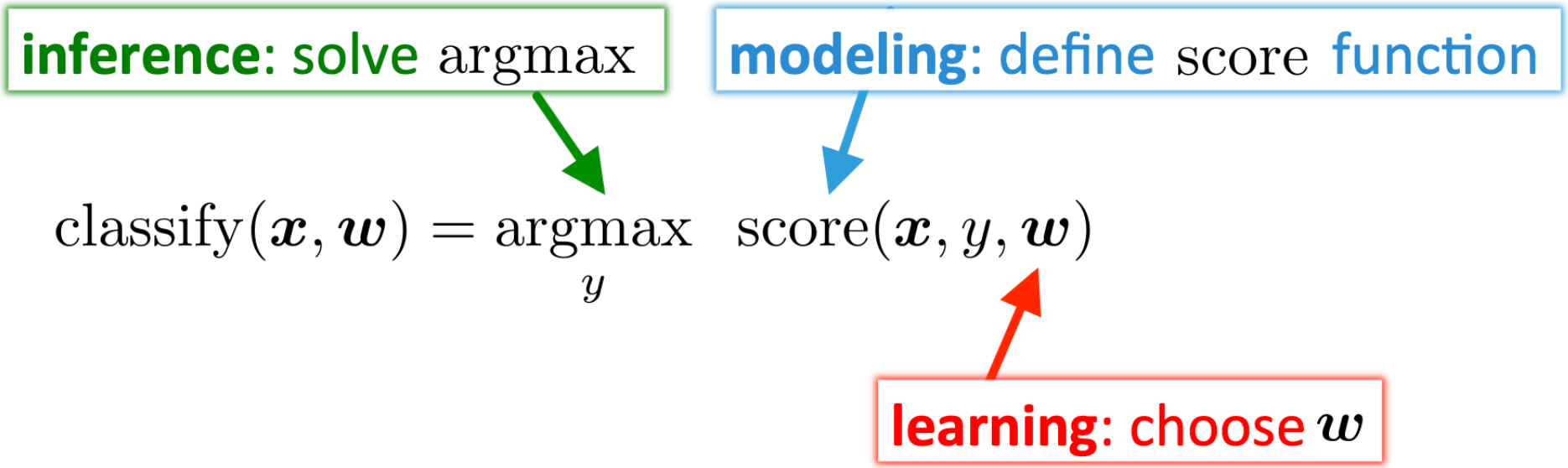
- **Modeling**: How do we assign a score to an (x,y) pair using parameters w?

Modeling, Inference, Learning



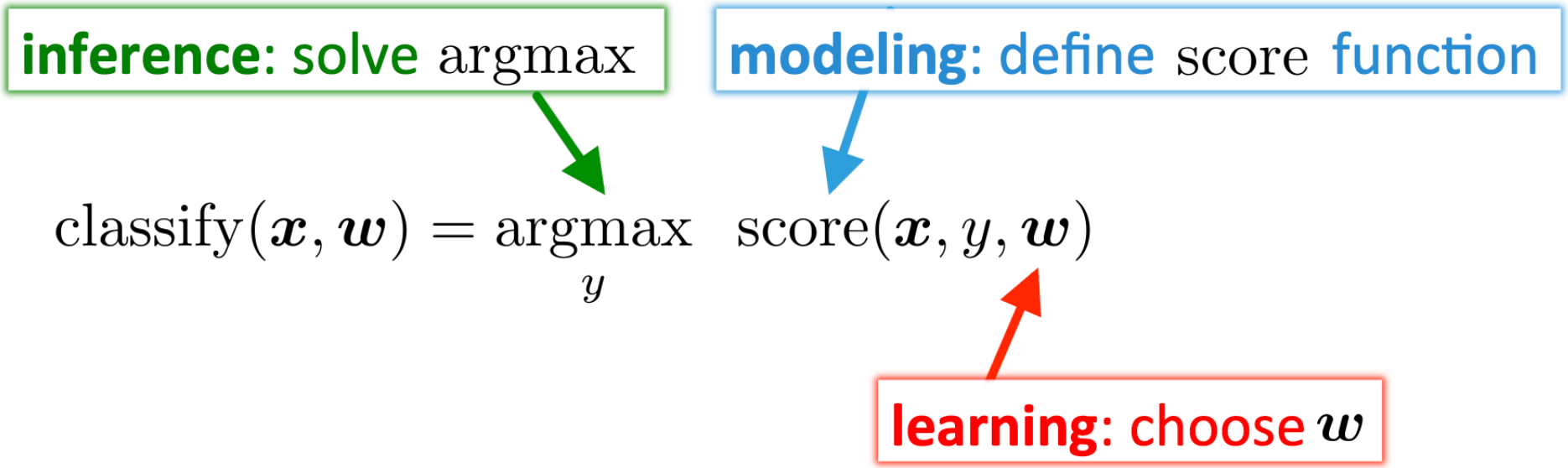
- **Inference**: How do we efficiently search over the space of all labels?

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

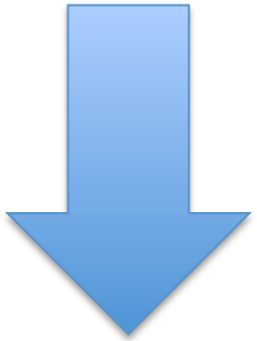
- Parameters are arranged in a vector **w**
- score function is linear in **w**:

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

- **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** ?

Features for NLP

- NLP datasets include inputs and outputs
- Features are usually not included
- You have to define your own features
- Contrast this with UCI datasets, which include a fixed-length **dense** feature vector for every instance
- In (traditional) NLP, features usually **sparse**

Defining Features

- This is a large part of NLP
- Last 30 years: feature engineering
- Last 10 years: representation learning

- Learning representations doesn't mean that we don't have to look at the data or the output!
- There's still plenty of engineering required in representation learning

Feature Engineering

- Often decried as “costly, hand-crafted, expensive, domain-specific”, etc.
- But in practice, simple features typically give the bulk of the performance

- Let’s get concrete: how should we define features for text classification?

N-gram

- **N-gram**: a sequence consisting of consecutive **n** words
- E.g., “Obama is going to”
  - unigram: “Obama”, “is”, “going”, “to”
  - bigram: “Obama is”, “is going”, “going to”
  - trigram: “Obama is going”, “is going to”

  …

Unigram Binary Features

- Two example features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1$ if S is true, 0 otherwise

- We usually think in terms of feature **templates**
- Unigram binary feature template:

$$f^{\text{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- To create features, this feature template is instantiated for particular labels and words

Feature Count Cutoffs

- Problem: some features are extremely rare
- Solution: only keep features that appear at least *k* times in the training data

Example: Part-of-Speech Tagging

- There are 45 part-of-speech (POS) tags in the Penn Treebank
- We don't want to create features for all 45*|V| combinations of tags and words
  - too many features to store in memory and too many feature weights to learn
- Most words appear with <= 3 unique POS tags in the training set
- So we use feature count cut-offs and only create features for combinations that appear enough times in the training data

Feature Count Cutoffs (Example)

- Training dataset with 2 examples:

great movie	positive
not so great	negative

- And a single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } word]$$

- What features would be in the model with a feature count cutoff of 2?

Feature Count Cutoffs (Example)

- Training dataset with 2 examples:

great movie	positive
not so great	negative

- And a single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } word]$$

- What features would be in the model with a feature count cutoff of 2?
  - none

Feature Count Cutoffs (Example)

- Training dataset with 2 examples:

great movie	positive
not so great	negative

- And a single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } word]$$

- What features would be in the model with a feature count cutoff of 1?

- Training dataset with 2 examples:

great movie	positive
not so great	negative

- And a single feature template:

$$f^{\text{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- What features would be in the model with a feature count cutoff of 1?

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{movie}]$$
$$f_3(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{not}]$$
$$f_4(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{so}]$$
$$f_5(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

- Training dataset with 2 examples:

great movie	positive
not so great	negative

- And a single feature template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } word]$$

- What additional features would be in the model with a feature count cutoff of 0?

$$f_6(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } movie]$$
$$f_7(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } not]$$
$$f_8(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } so]$$

Higher-Order Binary Feature Templates

- Unigram binary template:

$$f^{\mathrm{u,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{word}]$$

- Bigram binary template:

$$f^{\mathrm{b,b}}(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ label}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{“word1 word2”}]$$

- Trigram binary template:

…

Unigram **Count** Features

- A “count” feature returns the count of a particular word in the text

- Unigram count feature template:

$$f^{\mathrm{u,c}}(\boldsymbol{x}, y) = \begin{cases} \sum_{i=1}^{|\boldsymbol{x}|} \mathbb{I}[x_i = \ word], & \text{if } \mathbb{I}[y = \ \text{label}] \\ 0, & \text{otherwise} \end{cases}$$

Feature Engineering for Text Classification

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

- Two features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1$ if S is true, 0 otherwise

- What do you expect the weights to be?

$$w_1 > w_2? \qquad w_1 = w_2? \qquad w_1 < w_2?$$

Feature Engineering for Text Classification

- Two features:

$$f_1(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ positive}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$
$$f_2(\boldsymbol{x}, y) = \mathbb{I}[y = \text{ negative}] \wedge \mathbb{I}[\boldsymbol{x} \text{ contains } \textit{great}]$$

where $\mathbb{I}[S] = 1$ if S is true, 0 otherwise

- Let's say we set $w_1 > w_2$
- On sentences containing "***great***" in the Stanford Sentiment Treebank training data, this would get us an accuracy of 69%
- But "***great***'' only appears in 83/6911 examples

Feature Engineering for Text Classification

ambiguity: "***great***" may not mean positive sentiment

- On sentences containing "***great***" in the Stanford Sentiment Treebank training data, this would get us an accuracy of 69%
- But "***great***'' only appears in 83/6911 examples

variability: many other words can indicate positive sentiment

Feature Engineering

- Usually, great indicates positive sentiment:
  *The most wondrous love story in years, it is a **great** film.*

  *A **great** companion piece to other Napoleon films.*

- Sometimes not. Why?
  - **Negation**: It's not a great monster movie.
  - **Different sense**: There's a great deal of corny dialogue and preposterous moments.
  - **Multiple sentiments**: A great ensemble cast can't lift this heartfelt enterprise out of the familiar.

Text Classification

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

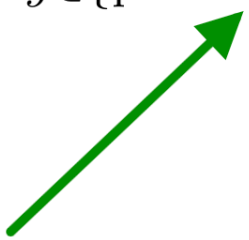
inference: solve argmax

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

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

Inference for Text Classification

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

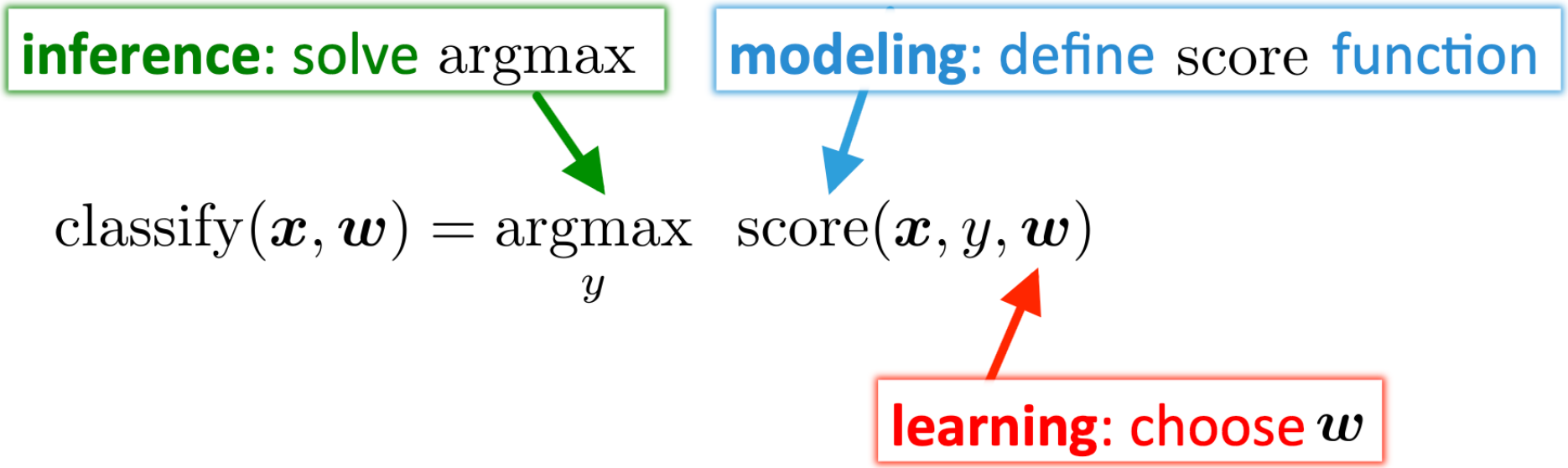


- trivial (loop over labels)

Text Classification

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

Modeling, Inference, Learning



- **Learning**: How should we choose values for the weights ***w***?

Text Classification

- Modeling
- Inference
- Learning
  - **empirical risk minimization**
  - surrogate loss functions
  - gradient-based optimization

Cost Functions

- Cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{R}_{\geq 0}$$

- Should be as close as possible to the actual evaluation metric for your task
- Usual conventions

$$\text{cost}(y, y) = 0$$
$$\text{cost}(y, y') = \text{cost}(y', y)$$

Cost Functions

- Cost function: scores outputs against a gold standard

$$\text{cost} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}_{\geq 0}$$

- Should be as close as possible to the actual evaluation metric for your task
- for classification, what cost should we use?

$$\text{cost}(y, y') = \mathbb{I}[y \neq y']$$

Risk Minimization

- given training data: $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$

  where each $y^{(i)} \in \mathcal{L}$ is a label

- assume data is drawn iid (independently and identically distributed) from (unknown) joint distribution $P(\boldsymbol{x}, y)$
- we want to solve the following:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}))\right]$$

problem: *P* is unknown

Empirical Risk Minimization (Vapnik et al.)

- Replace expectation with sum over examples:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

problem: NP-hard even for binary classification with linear models

- Solution: replace “cost loss” (also called “0-1” loss) with a surrogate function that is easier to optimize

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{cost}(y^{(i)}, \operatorname{classify}(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

generalize to permit any loss function

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

- Cost loss / 0-1 loss: $\operatorname{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w}))$

Text Classification

- Modeling
- Inference
- Learning
  - empirical risk minimization
  - **surrogate loss functions**
  - gradient-based optimization

Surrogate Loss Functions

- Cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$
  - why is this so difficult to optimize? not necessarily continuous, can't use gradient-based optimization
- max-score loss: $\text{loss}_{\text{maxscore}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w})$
- perceptron loss: $\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$
  - loss function underlying perceptron algorithm (Rosenblatt, 1957-58)

Surrogate Loss Functions

- Cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$
- perceptron loss: $\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$
- hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

  - loss function underlying support vector machines

Surrogate Loss Functions

- Cost loss / 0-1 loss: $\text{loss}_{\text{cost}}(\boldsymbol{x}, y, \boldsymbol{w}) = \text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w}))$
- perceptron loss: $\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$

- hinge loss:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \text{cost}(y, y'))$$

- hinge loss for our classification setting:

$$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} (\text{score}(\boldsymbol{x}, y', \boldsymbol{w}) + \delta \mathbb{I}(y \neq y'))$$

tunable hyperparameter

Log Loss

$$\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x})$$

- Minimize negative log of conditional probability of output given input
  - sometimes called “maximizing conditional likelihood”
- But we don’t have a probabilistic model, we just have a score function

Score → Probability

- Can turn score into probability by exponentiating (to make it positive) and normalizing:

$$p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \propto \exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}$$

$$p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) = \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}}$$

- This is open called a “softmax” function

Log Loss

$$\begin{aligned}
\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) &= -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \\
&= -\log \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}} \\
&= -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}
\end{aligned}$$

- Similar to perceptron loss!
- Replace max with "softmax" (a different kind of softmax)

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

Log Loss

$$\begin{aligned}
\text{loss}_{\log}(\boldsymbol{x}, y, \boldsymbol{w}) &= -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x}) \\
&= -\log \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}} \\
&= -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}
\end{aligned}$$

- Log loss is used in:
  - logistic regression classifiers,
  - conditional random fields,
  - maximum entropy (“maxent”) models

Log Loss

$$\text{loss}_{\text{log}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\log p_{\boldsymbol{w}}(y \mid \boldsymbol{x})$$

$$= -\log \frac{\exp\{\text{score}(\boldsymbol{x}, y, \boldsymbol{w})\}}{\sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}}$$

$$= -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \log \sum_{y' \in \mathcal{L}} \exp\{\text{score}(\boldsymbol{x}, y', \boldsymbol{w})\}$$

- Issue: can be very expensive due to summation over all possible outputs!

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{w}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{w}) + \max_{y' \in \mathcal{L}} \text{score}(\boldsymbol{x}, y', \boldsymbol{w})$$

Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w})$$

Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\operatorname{cost}(y, \operatorname{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \; \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Regularized Empirical Risk Minimization

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \mathbb{E}_{P(\boldsymbol{x},y)} \left[\text{cost}(y, \text{classify}(\boldsymbol{x}, \boldsymbol{w})) \right]$$

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \; \sum_{i=1}^{|\mathcal{T}|} \text{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

regularization strength

regularization term

Regularization Terms

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{|\mathcal{T}|} \operatorname{loss}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

- Most common: penalize large parameter values
- Intuition: large parameters might be instances of overfitting
- Examples:
  - L_2 **regularization**: (also called Tikhonov regularization or ridge regression)
  - L_1 **regularization**: (also called basis pursuit or LASSO)

Regularization Terms

- **L_2 regularization**: $R_{\mathrm{L2}}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2$
  - differentiable, widely-used
- **L_1 regularization:** $R_{\mathrm{L1}}(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_i |w_i|$
  - not differentiable (but is subdifferentiable)
  - leads to sparse solutions (many parameters become zero!)

Experimental Practice

- In the beginning, we just had **data**
- First approach: split into **train** and **test**
  - motivation: simulate conditions of applying system in practice

- But, there's a problem with this...
  - we need to explore and evaluate methodological choices
  - after multiple evaluations on **test**, it is no longer a simulation of real-world conditions

Experimental Practice

- We need to explore/evaluate methodological choices
- Second approach: divide data into **train**, **test**, and a third set called development (**dev**) or validation (**val**)
  - use **dev**/**val** to evaluate choices
  - then, when ready to write the paper, evaluate the best model on **test**
- But, there's still a problem with this...
  - overfittng to **dev**/**val**