CIS 530: Text Processing Wrap up & Logistic Regression

MONDAYS AND WEDNESDAYS 1:30-3PM
~~3401 WALNUT, ROOM 401B~~ ANNENBERG 110
[COMPUTATIONAL-LINGUISTICS-CLASS.ORG](http://computational-linguistics-class.org)

PROFESSOR CALLISON-BURCH

Reminders

QUIZ 1 IS DUE TONIGHT BEFORE 11:59PM.

HELP US HELP YOU ON PIAZZA

READ TEXTBOOK CHAPTER 5

Wrap-up: Text Processing

READ: JURAFSKY AND MARTIN CHAPTER 2

WORD NORMALIZATION AND STEMMING

Recap: Normalization

Need to "normalize" terms

- Information Retrieval: indexed text & query terms must have same form.
  - We want to match ***U.S.A.*** and ***USA***

We implicitly define equivalence classes of terms

- e.g., deleting periods in a term

Alternative: asymmetric expansion:

- Enter: ***window*** Search: ***window, windows***
- Enter: ***windows*** Search: ***Windows, windows, window***
- Enter: ***Windows*** Search: ***Windows***

Potentially more powerful, but less efficient

Issues in Tokenization

Finland's capital → Finland Finlands Finland's ?

what're, I'm, isn't → What are, I am, is not

Hewlett-Packard → Hewlett Packard ?

state-of-the-art → state of the art ?

Lowercase → lower-case lowercase lower case ?

San Francisco → one token or two?

m.p.h., PhD. → ??

French

- ***L'ensemble*** → one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***

German noun compounds are not segmented

- ***Lebensversicherungsgesellschaftsangestellter***
- 'life insurance company employee'
- German information retrieval needs **compound splitter**

Tokenization: language issues

Chinese and Japanese no spaces between words:

- 莎拉波娃现在居住在美国东南部的佛罗里达。
- 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Sharapova now lives in US southeastern Florida

Further complicated in Japanese, with multiple alphabets intermingled

- Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romaji

End-user can express query entirely in hiragana!

Tokenization: language issues

Word Tokenization in Chinese

Also called **Word Segmentation**

Chinese words are composed of characters

- Characters are generally 1 syllable and 1 morpheme.
- Average word is 2.4 characters long.

Standard baseline segmentation algorithm:

- Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation Algorithm

Given a wordlist of Chinese, and a string:

1) Start a pointer at the beginning of the string
2) Find the longest word in dictionary that matches the string starting at pointer
3) Move the pointer over the word in string
4) Go to 2

Max-match segmentation illustration

Thecatinthehat

Thetabledownthere

the cat in the hat

the table down there

theta bled own there

Doesn’t generally work in English!

But works surprisingly well in Chinese

- 莎拉波娃现在居住在美国东南部的佛罗里达。
- 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

Modern probabilistic segmentation algorithms even better

Byte-Pair Encoding for Tokenization

Modern tokenizers use data to automatically determine what size tokens we should use, rather than relying on whitespace, or max-matc

Sometimes we want a space delimited words like *spinach* to be a token

Other times we might want **multi-word** units like *New York Times*

Sometimes we want **subword** units like morphemes *–est* or *–er*

Subword units are helpful for dealing with unknown words.

Byte-Pair Encoding for Tokenization

The BPE algorithm tokenizes text, such that most tokens are words, but some tokens are frequent morphemes or other subwords like *–er.*

Unseen word can be represented by combining the parts.

BPE was originally used for text compression, but was repurposed for tokenization in 2016 by Rico Sennrich, Barry Haddow, and Alexanra Birch to translate rare and unseen words.

Start with a set of symbols that is the set of characters, plus an end of word symbol.

Byte-Pair Encoding for Tokenization

Start with a set of symbols that is the set of characters, plus an end of word symbol.

At each step, count the number of symbol pairs, find the most frequent pair (‘A’, ‘B’), and replace it with the new merged symbol (‘AB’).

Repeat this merge step *k* times.

The resulting symbol set will consist of the original characters plus *k* new symbols.

```python
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e s t </w>' : 2,
    'n e w e r </w>':6, 'w i d e r </w>':3, 'n e w </w>':2}
num_merges = 8

for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

	dictionary	**vocabulary**
5	`l o w _`	`_, d, e, i, l, n, o, r, s, t, w`
2	`l o w e s t _`	
6	`n e w e r _`	
3	`w i d e r _`	
2	`n e w _`	

	dictionary	**vocabulary**
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

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

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

	dictionary	vocabulary
5	`l o w _`	`_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew`
2	`l o w e s t _`	
6	`n ew er_`	
3	`w i d er_`	
2	`n ew _`	

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o’	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

Basic Text Processing

WORD NORMALIZATION AND STEMMING

Normalization

Need to "normalize" terms

- Information Retrieval: indexed text & query terms must have same form.
  - We want to match ***U.S.A.*** and ***USA***

We implicitly define equivalence classes of terms

- e.g., deleting periods in a term

Alternative: asymmetric expansion:

- Enter: ***window*** Search: ***window, windows***
- Enter: ***windows*** Search: ***Windows, windows, window***
- Enter: ***Windows*** Search: ***Windows***

Potentially more powerful, but less efficient

Case folding

Applications like IR: reduce all letters to lower case

- Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., ***General Motors***
  - ***Fed*** vs. ***fed***
  - ***SAIL*** vs. ***sail***

For sentiment analysis, MT, Information extraction

- Case is helpful (***US*** versus ***us*** is important)

Lemmatization

Reduce inflections or variant forms to base form

- *am, are, is* $\rightarrow$ *be*
- *car, cars, car's, cars'* $\rightarrow$ *car*

the boy's cars are different colors $\rightarrow$ *the boy car be different color*

Lemmatization: have to find correct dictionary headword form

Machine translation

- Spanish quiero (‘I want’), quieres (‘you want’) same lemma as querer ‘want’

Morphology

Morphemes:

- The small meaningful units that make up words
- **Stems**: The core meaning-bearing units
- **Affixes**: Bits and pieces that adhere to stems
- Often with grammatical functions

Stemming

Reduce terms to their stems in information retrieval

Stemming is crude chopping of affixes

- language dependent
- e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm
The most common English stemmer

Step 1a

```
sses → ss    caresses → caress
ies  → i     ponies   → poni
ss   → ss    caress   → caress
s    → ø     cats     → cat
```

Step 1b

```
(*v*)ing → ø  walking   → walk
              sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

Step 2 (for long stems)

```
ational→ ate  relational→ relate
izer→ ize     digitizer → digitize
ator→ ate     operator  → operate
…
```

Step 3 (for longer stems)

```
al   → ø  revival    → reviv
able → ø  adjustable → adjust
ate  → ø  activate   → activ
…
```

`(*v*)ing` → ø	walking	→	walk
	sing	→	sing

```
(*v*)ing → ø  walking  → walk
              sing     → sing
```

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort –nr
```

1312 King	548 being
548 being	541 nothing
541 nothing	152 something
388 king	145 coming
375 bring	130 morning
358 thing	122 having
307 ring	120 living
152 something	117 loving
145 coming	116 Being
130 morning	102 going

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort –nr
```

Some languages requires complex morpheme segmentation

- Turkish
- Uygarlastiramadiklarimizdanmissinizcasina
- `(behaving) as if you are among those whom we could not civilize’
- Uygar `civilized’ + las `become’

  + tir `cause’ + ama `not able’

  + dik `past’ + lar ‘plural’

  + imiz ‘p1pl’ + dan ‘abl’

  + mis ‘past’ + siniz ‘2pl’ + casina ‘as if’

Dealing with complex morphology is sometimes necessary

Basic Text Processing

SENTENCE SEGMENTATION AND DECISION TREES

Sentence Segmentation

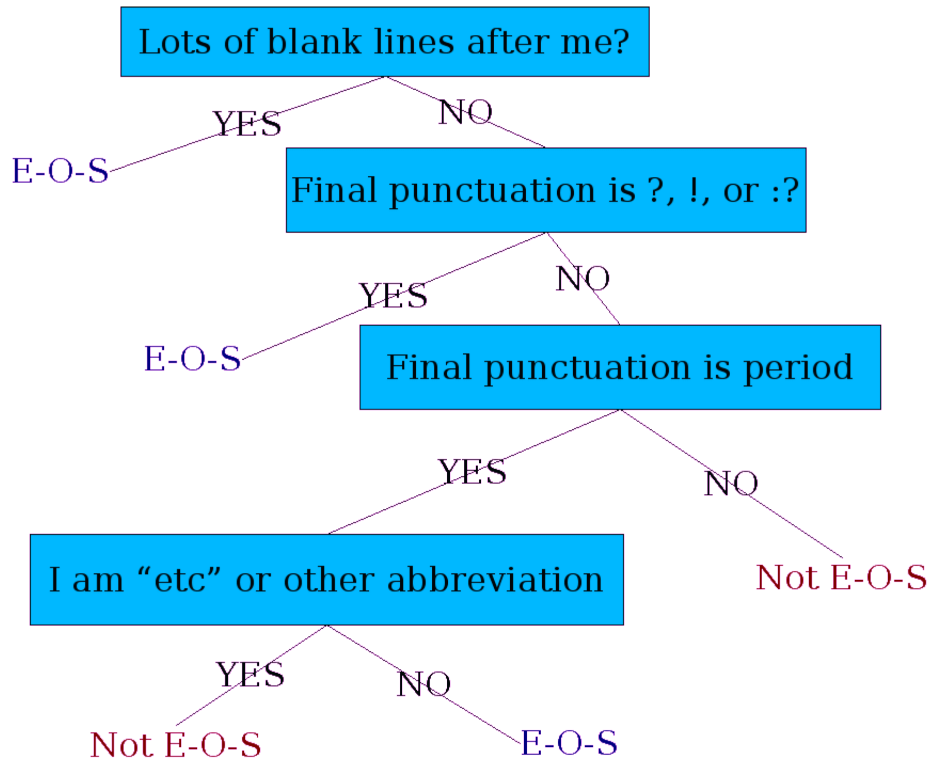
!, ? are relatively unambiguous

Period "." is quite ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Build a binary classifier

- Looks at a "."
- Decides EndOfSentence/NotEndOfSentence
- Classifiers: hand-written rules, regular expressions, or machine-learning



Determining if a word is end-of-sentence: a Decision Tree

Case of word with ".": Upper, Lower, Cap, Number

Case of word after ".": Upper, Lower, Cap, Number

Numeric features

- Length of word with "."
- Probability(word with "." occurs at end-of-s)
- Probability(word after "." occurs at beginning-of-s)

More sophisticated decision tree features

A decision tree is just an if-then-else statement

The interesting research is choosing the features

Setting up the structure is often too hard to do by hand

- Hand-building only possible for very simple features, domains
  - For numeric features, it’s too hard to pick each threshold
- Instead, structure usually learned by machine learning from a training corpus

Implementing Decision Trees

We can think of the questions in a decision tree

As features that could be exploited by any kind of classifier

- Logistic regression
- SVM
- Neural Nets
- etc.

Decision Trees and other classifiers

Logistic Regression

JURAFSKY AND MARTIN CHAPTER 5

Generative v. Discriminative Classifiers and cats v. dogs

Naive Bayes is a **generative** classifier

Logistic regression is a is a **discriminiative classifier**

Tanks v. no tanks

A (possibly apocryphal) tale in artificial intelligence tells about researchers training a neural network to detect tanks in photographs for a DARPA project.

They apparently succeed. Great let's deploy it!
Oops! It didn't work as well as we thought it would.

Later they realized the photographs had been collected under specific conditions for tanks/non-tanks and the classifier had simply learned to distinguish between the time of day.

Generative v. Discriminative Classifiers

Naïve Bayes doesn't directly compute P(c|d). Instead it computes it using two terms:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overbrace{P(d|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}}$$

A **generative model** uses the likelihood term, which expresses how to generate the features of a document *if we knew it was of class c*.

A **discriminative model** attempts to directly compute P(c|d).
It may learn to assign a **high weight** to document features that directly improve its **ability to discriminate between classes**

Unlike the generative model, good paramters estimates for a discriminative model don't help it generate an example of one of the classes.

Classifier components

1. A **feature representation** of the input.
2. A classification function that computes $\hat{y}$, estimated class via $p(y|x)$. Logistic regression will use **sigmoid** and **softmax**
3. An objective function used during learning to minimize error on the training examples. We will discuss **cross-entropy loss**.
4. An algorithm for optimizing the objective function like **stochastic gradient descent**.

Sentiment classifier

Input: "Spiraling away from narrative control as its first three episodes unreel, this series, about a post-apocalyptic future in which nearly everyone is blind, wastes the time of Jason Momoa and Alfre Woodard, among others, on a story that starts from a position of fun, giddy strangeness and drags itself forward at a lugubrious pace."

Output: positive (1) or **negative (0)**

Sentiment classifier

For sentiment classification, consider an input observation x, represented by a vector of **features** $[x_1, x_2, \ldots, x_n]$. The classifier output y can be 1 (positive sentiment) or 0 (negative sentiment). We want to estimate $\boldsymbol{P(y = 1|x)}$.

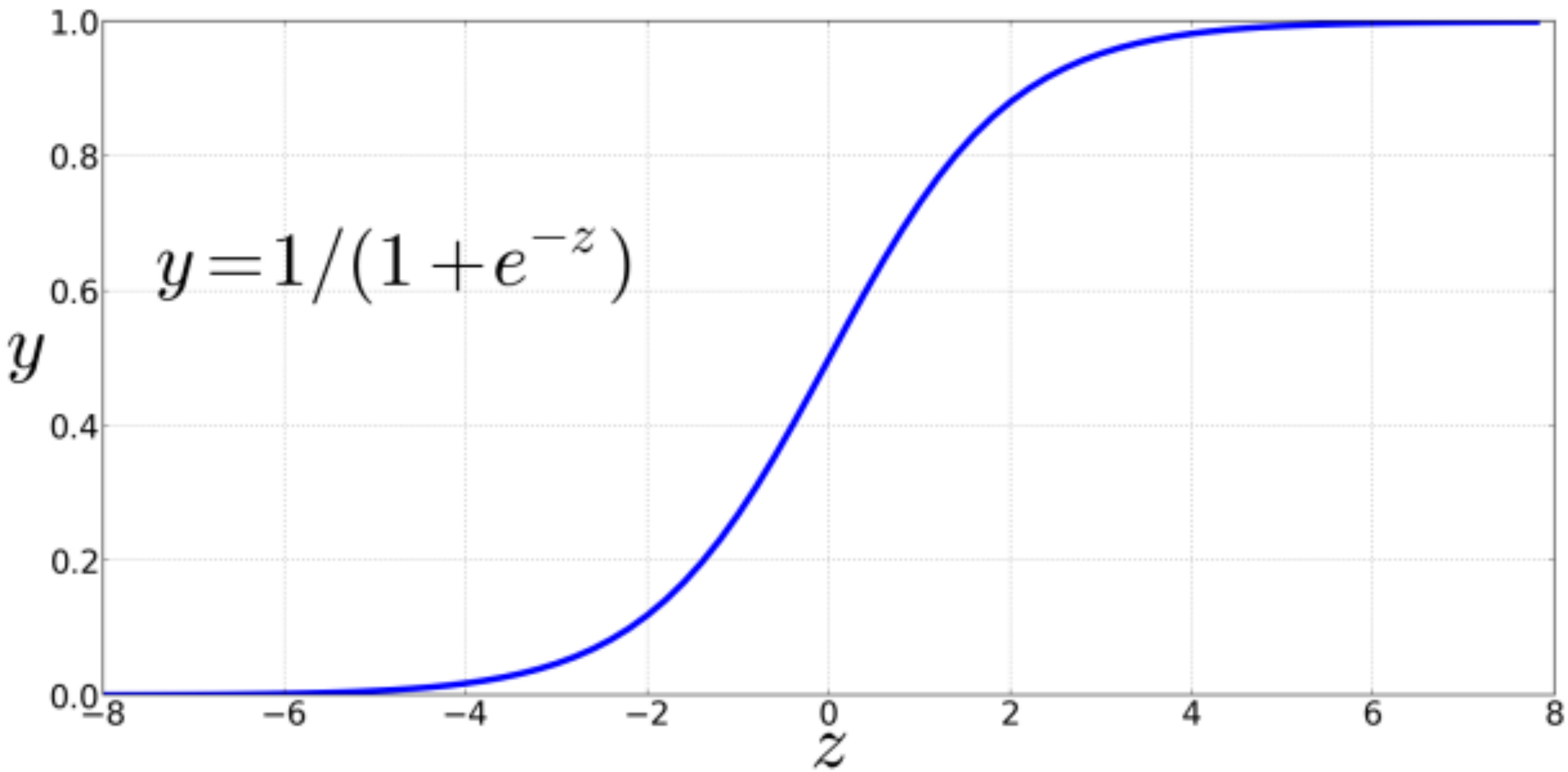
Logistic regression solves this task by learning, from a training set, a vector of **weights** and a **bias term**.

$$z = \sum_i w_i x_i + b$$

We can also write this as a dot product:

$$z = w \cdot x + b$$

Sigmoid function



Probabilities

$$P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

Decision boundary

Now we have an algorithm that given an instance *x* computes the probability $P(y = 1|x)$. How do we make a decision?

$$\hat{y} = \begin{cases} 1 \; if \; P(y = 1|x) > 0.5 \\ 0 \; otherwise \end{cases}$$

For a test instance *x*, we say **yes** if the probability $P(y = 1|x)$ is more than .5, and **no** otherwise. We call .5 the decision boundary

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value
x_1	Count of positive lexicon words	
x_2	Count of negative lexicon words	
x_3	Does no appear? (binary feature)	
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	
x_3	Does no appear? (binary feature)	
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	
x_4	Number of 1st and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1^{st} and 2nd person pronouns	3
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Word count = 64, ln(64) = 4.15

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1st and 2nd person pronouns	3
x_5	Does ! appear? (binary feature)	0
x_6	Log of the word count for the document	4.15

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	
x_2	Count of negative lexicon words	2	-5.0	
x_3	Does no appear? (binary feature)	1	-1.2	
x_4	Num 1st and 2nd person pronouns	3	0.5	
x_5	Does ! appear? (binary feature)	0	2.0	
x_6	Log of the word count for the doc	4.15	0.7	
b	bias	1	0.1	

$$z = \sum_i w_i x_i + b$$

Computing Z

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	7.5
x_2	Count of negative lexicon words	2	-5.0	-10
x_3	Does no appear? (binary feature)	1	-1.2	-1.2
x_4	Num 1^{st} and 2nd person pronouns	3	0.5	1.5
x_5	Does ! appear? (binary feature)	0	2.0	0
x_6	Log of the word count for the doc	4.15	0.7	2.905
b	bias	1	0.1	.1

$$z = \sum_i w_i x_i + b$$

z=0.805

Sigmoid(Z)

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	7.5
x_2	Count of negative lexicon words	2	-5.0	-10
x_3	Does no appear? (binary feature)	1	-1.2	-1.2
x_4	Num 1st and 2nd person pronouns	3	0.5	1.5
x_5	Does ! appear? (binary feature)	0	2.0	0
x_6	Log of the [illegible]	[illegible]	0.7	2.905
b	bias	[illegible]	0.1	.1

$\sigma(0.805)$
$= 0.69$

Learning in logistic regression

How do we get the weights of the model? We learn the parameters (weights + bias) via learning. This requires 2 components:

1. An objective function or **loss function** that tells us *distance* between the system output and the gold output. We will use **cross-entropy loss**.
2. An algorithm for optimizing the objective function. We will use stochastic gradient descent to **minimize** the **loss function.**

Loss functions

We need to determine for some observation *x* how close the classifier output ($\hat{y}$= σ ($w \cdot x + b$)) is to the correct output (y, which is 0 or 1).

$$L(\hat{y}, y) = \text{how much } \hat{y} \text{ differs from the true } y$$

One example is mean squared error

$$L_{MSE}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Loss functions for probabilistic classification

We use a loss function that prefers the correct class labels of the training example to be more likely.

Conditional maximum likelihood estimation: Choose parameters *w, b* that maximize the (log) probabilities of the true labels in the training data.

The resulting loss function is the negative log likelihood loss, more commonly called the **cross entropy loss.**

Loss functions for probabilistic classification

For one observation x, let's **maximize** the probability of the correct label p(y|x).

$$p(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$$

If y = 1, then $\mathrm{p(y|x)} = \hat{y}$.

If y = 0, then $\mathrm{p(y|x)} = 1 - \hat{y}$.

Loss functions for probabilistic classification

Change to logs (still maximizing)

$$\log p(y|x) = \log[\hat{y}^y (1-\hat{y})^{1-y}]$$

$$= y \log \hat{y} + (1-y) \log(1-\hat{y})$$

This tells us what log likelihood should be maximized. But for loss functions, we want to minimize things, so we'll flip the sign.

Cross-entropy loss

The result is cross-entropy loss:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Finally, plug in the definition for $\hat{\boldsymbol{y}}$= σ $(w \cdot x) + b$

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) \ + (1-y)\log(1-\sigma(w \cdot x + b))]$$

Cross-entropy loss

Why does minimizing this negative log probability do what we want? We want the **loss** to be **smaller** if the model's estimate is **close to correct**, and we want the **loss** to be **bigger if it is confused.**

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

P(sentiment=1|It's hokey...) = 0.69. Let's say y=1.

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y) \log(1 - \sigma(w \cdot x + b))]$$

$$= -[\log \sigma(w \cdot x + b)]$$

$$= -\log(0.69) = \mathbf{0.37}$$

Cross-entropy loss

Why does minimizing this negative log probability do what we want? We want the **loss** to be **smaller** if the model's estimate is **close to correct**, and we want the **loss** to be **bigger if it is confused.**

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

P(sentiment=1|It's hokey...) = 0.69.   Let's **pretend** y=0.

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

$$= -[\log(1 - \sigma(w \cdot x + b))]$$

$$= -\log(0.31) = \mathbf{1.17}$$

Cross-entropy loss

Why does minimizing this negative log probability do what we want? We want the **loss** to be **smaller** if the model's estimate is **close to correct**, and we want the **loss** to be **bigger if it is confused.**

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

If our prediction is **correct**, then our CE loss is **lower**

$$= -\log(0.69) = \mathbf{0.37}$$

If our prediction is **incorrect**, then our CE loss is **higher**

$$-\log(0.31) = \mathbf{1.17}$$

Loss on all training examples

$$\log p(training\ labels) = \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$$

$$= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)})$$

$$= -\sum_{i=1}^{m} \mathrm{L_{CE}}(\hat{y}^{(i)}|y^{(i)})$$

Finding good parameters

We use gradient descent to find good settings for our weights and bias by minimizing the loss function.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters θ) the function's slope is rising the most steeply, and moving in the opposite direction.

Gradient descent

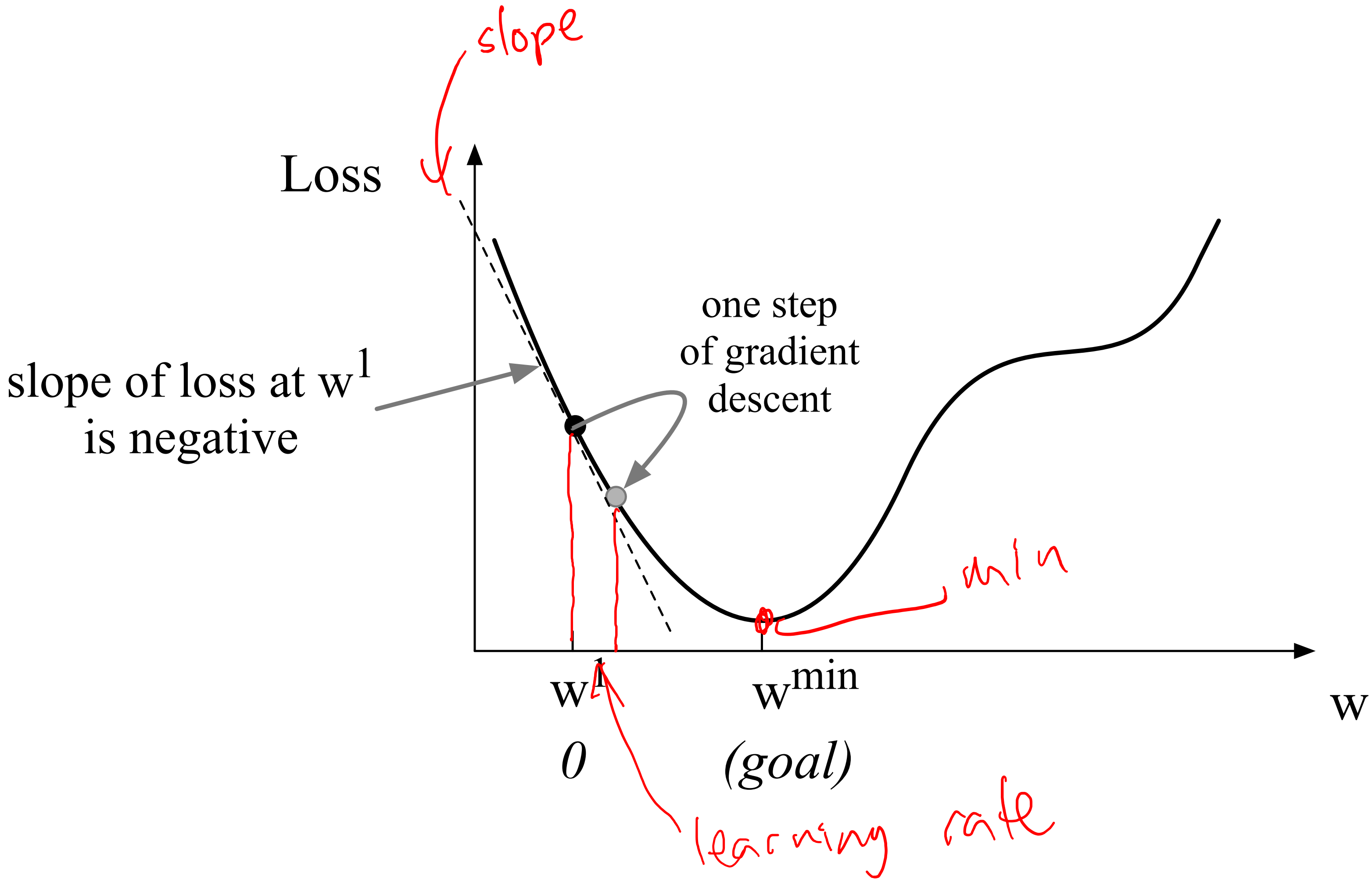
Global v. Local Minimums

For logistic regression, this loss function is conveniently **convex**.

A convex function has just **one minimum**, so there are no local minima to get stuck in.

So gradient descent starting from any point is guaranteed to find the minimum.

Iteratively find minimum



How much should we update the parameter by?

The magnitude of the amount to move in gradient descent is the value of the slope weighted by a learning rate η.

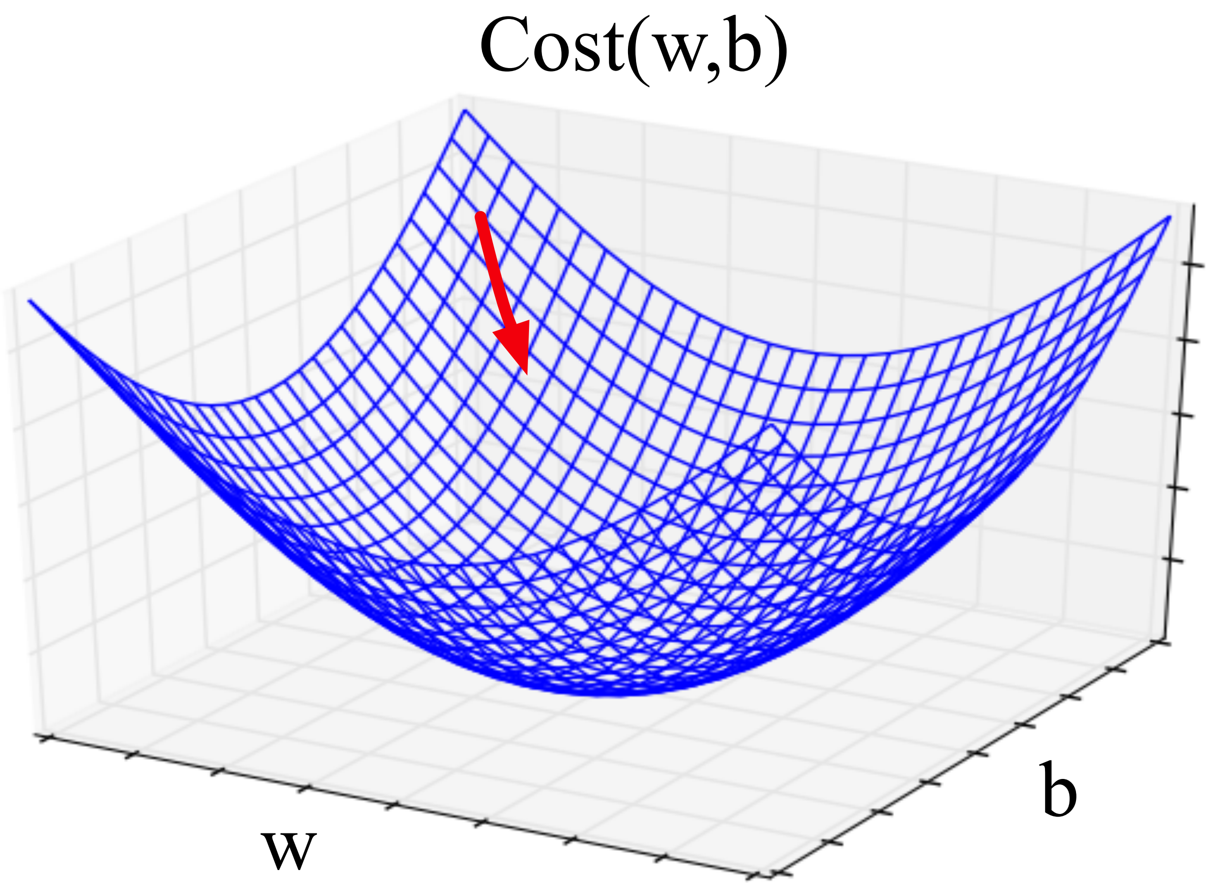
A higher/faster learning rate means that we should move *w* more on each step.

time t+1   "eta"   slope

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x;w)$$

new weight   old weight   minus   learning rate * derivative of of fn.

Many dimensions



Updating each dimension w_i

$$\nabla_\theta L(\underbrace{f(x;\theta)}_{\hat{y}}, y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta), y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta), y) \end{bmatrix}$$

our model's prediction for input x given parameters θ

The final equation for updating θ based on the gradient is

Learning rate

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

The Gradient

To update θ, we need a definition for the gradient $\nabla L(f(x;\theta), y)$.

For logistic regression the cross-entropy loss function is:

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

The derivative of this function for one observation vector x for a single weight w_j is

truth

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\underbrace{\sigma(w\cdot x+b)}_{\text{our model's prediction}}-y]x_j$$

⇐ * value of feature j

The gradient is a very intuitive value: the difference between the true y and our estimate for x, multiplied by the corresponding input value x_j .

Average Loss

$$Cost(w,b) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma\left(w \cdot x^{(i)} + b\right) + \left(1 - y^{(i)}\right)\log(1 - \sigma\left(w \cdot x^{(i)} + b\right))$$

This is what we want to minimize!!

The Gradient

The loss for a batch of data or an entire dataset is just the average loss over the *m* examples

$$Cost(w, b) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma\left(w \cdot x^{(i)} + b\right) + \left(1 - y^{(i)}\right) \log(1 - \sigma\left(w \cdot x^{(i)} + b\right))$$

The gradient for multiple data points is the sum of the individual gradients:

$$\frac{\partial Cost(w, b)}{\partial w_j} = \sum_{i=1}^{m} [\sigma\left(w \cdot x^{(i)} + b\right) - y^{(i)}] x_j^{(i)}$$

Stochastic gradient descent algorithm

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat T times

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$ # What is our estimated output $\hat{y}$?

Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?

$g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$ # How should we move θ to maximize loss ?

$\theta \leftarrow \theta - \eta\, g$ # go the other way instead

return θ

Worked example

Let's walk though a single step of the gradient descent algorithm. We'll use a simple sentiment classifier with just 2 features, and 1 training instance where the correct value is $y = 1$ (this is a positive review).

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of positive negative words)

The initial weights and bias in θ^0 are all set to 0, and the initial learning rate η is 0.1:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

The single update step requires that we compute the gradient, multiplied by the learning rate:

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

Worked example

The derivative of this function for **a single training example** *x* for a single weight w_j is

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [s(w \cdot x + b) - y]x_j$$

The gradient vector has 3 dimensions, for w_1, w_2, and b.
For our input, $x_1 = 3$ and $x_2 = 2$

$x_2 = 2$

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot\ x + b) - y)x_1 \\ (\sigma(w \cdot\ x + b) - y)x_2 \\ \sigma(w \cdot\ x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Worked example

Now that we have a gradient $\nabla_{w,b}$, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

So after one step of gradient descent, the weights have shifted to be:

w_1 = 0.15, w_2 = 0.1, and b= .05

Mini-batch training

Stochastic gradient descent chooses a **single random example** at a time, and updates its weights on that example. As a result the updates can fluctuate.

An alternate is **batch training,** which computes the gradient over the **entire dataset**. This gives a much better estimate of which direction to move the weights, but takes a long time to compute.

A commonly used compromise is **mini-batch training**, where we train on a small batch. The batch size can be 512 or 1024, often selected based on computational resources, so that all examples in the mini-batch can be processed in parallel. The loss is then accumulated.

Regularization

Overfitting is a problem with many machine learning models. Overfitting results in poor generalization and poor performance on unseen test set.

In logistic regression, if a feature only occurs in one class then it will get a **high weight**. Sometimes we are just modelling noisy factors that just accidentally correlate with the class.

Regularization is a way to penalize large weights. A regularization term is added to the loss function.

Lasso regression uses L1 regularization
Ridge regression uses L2 regularization

Multinomial logistic regression

Instead of binary classification, we often want more than two classes. For sentiment classification we might extend the class labels to be **positive, negative**, and **neutral**.

We want to know the probability of y for each class $c \in C$, $p(y = c|x)$.

To get a proper probability, we will use a **generalization of the sigmoid function** called the **softmax function.**

$$\text{softmax}(z_i) = \frac{e^{z_j}}{\sum_{j=1}^{k} e^{z_j}} \; 1 \leq i \leq k$$

Softmax

The softmax function takes in an input vector $z = [z_1, z_2, ..., z_k]$ and outputs a vector of values normalized into probabilities.

$$\text{softmax}(z) = [\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \cdots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}}]$$

For example, for this input:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

Softmax will output:

[0.056, 0.090, 0.007, 0.099, 0.74, 0.010]

Next time: Neural Nets