Lecture 5: Naive Bayes Classification, Features, Linear Models, Perceptron

USC VSoE CSCI 544: Applied Natural Language Processing

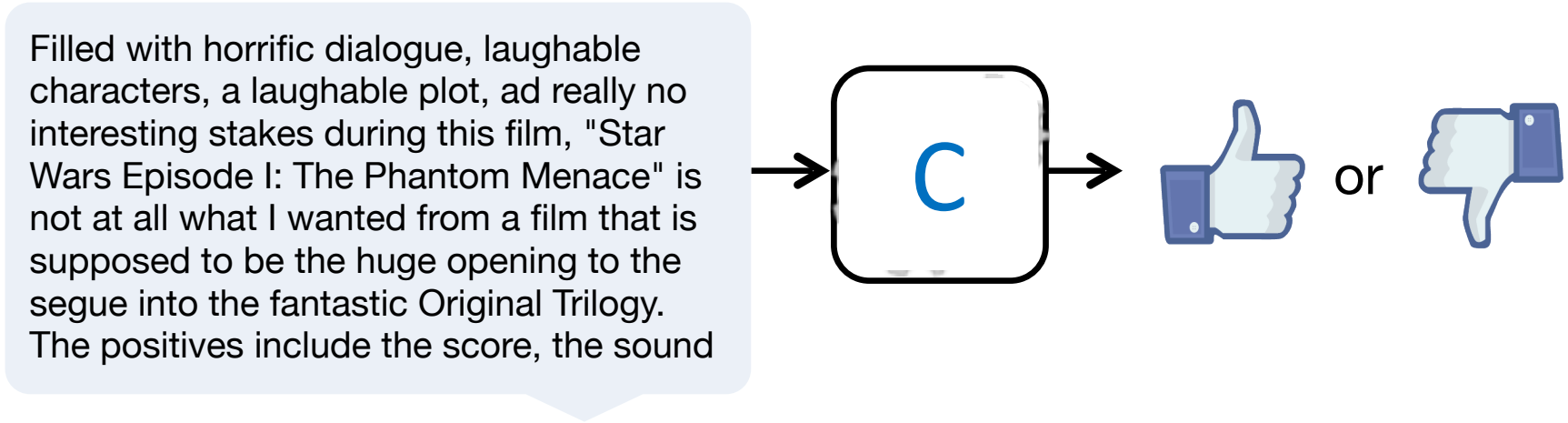
Jonathan May -- 梅約納

September 6, 2017

based on Nathan Schneider's slides

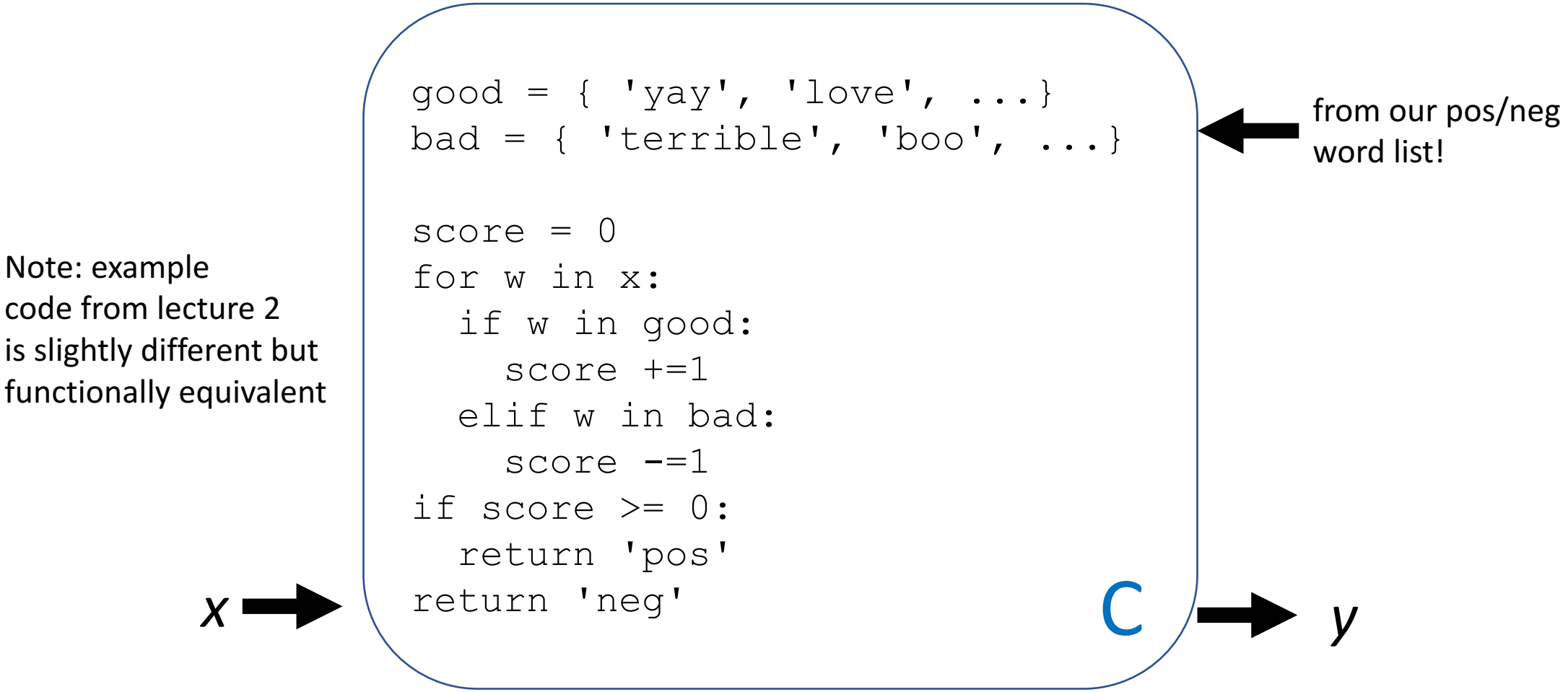
Sentiment Analysis

- Recall the task:



- This is a **classification** task: our input is free text but our output is a fixed set of labels
- In this lecture, input/observed data is denoted *x* (or set *X*) and output/prediction is *y* (or set *Y*)

Our Previous Rule-Based Classifier

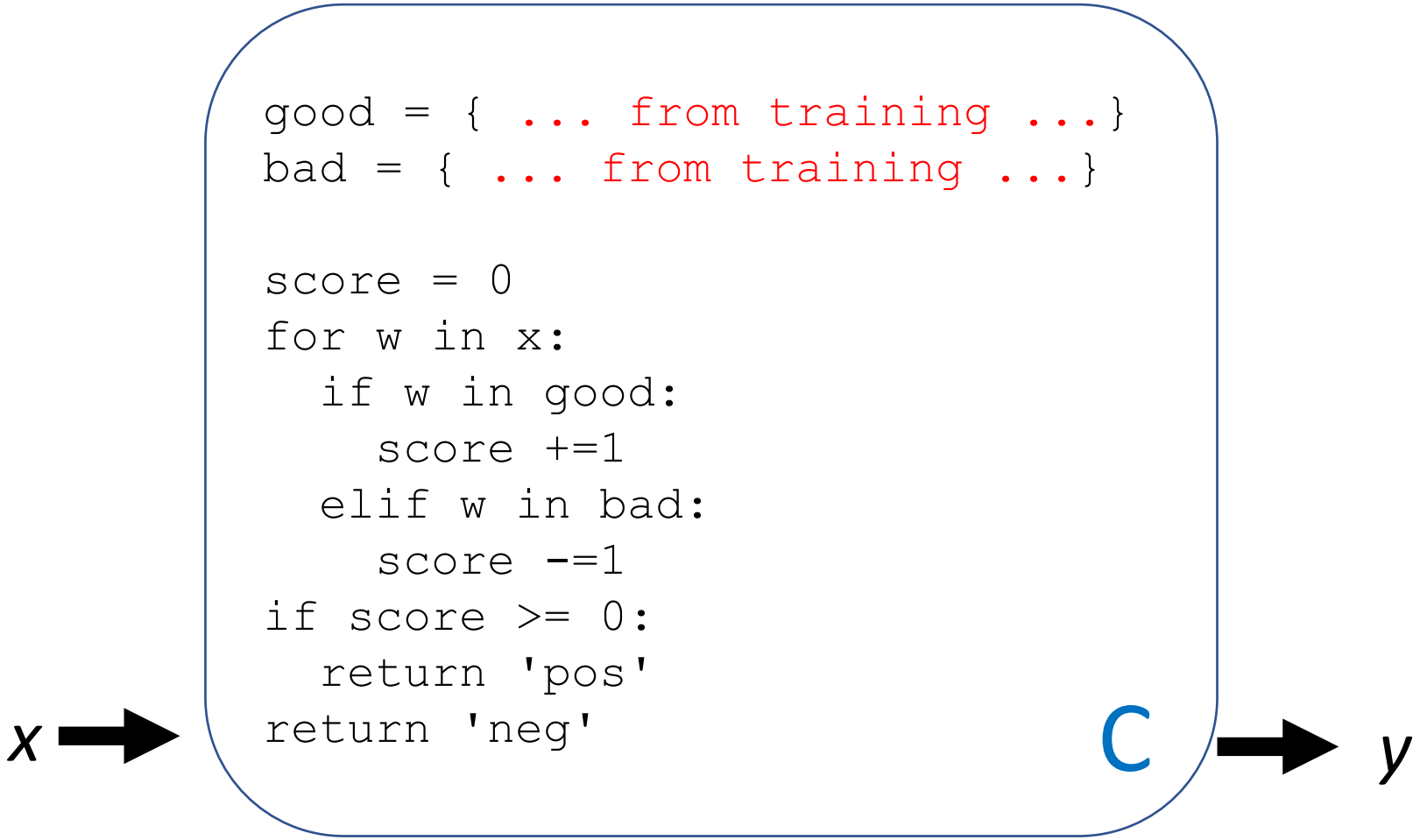


Supervised Classification

- Rather than top-down features, let's use data-driven features
  - Our intuitions about word sentiment aren't perfect
- **Supervised** (aka Inductive) = learn from **labeled** examples. Recipe:
  - **training** corpus of (*x, y*) (review, label) pairs
  - learning algorithm
- Other kinds of learning (not covered here):
  - **Unsupervised**: Data is provided but no labels are given
  - **Semi-Supervised:** Data is provided but only some of it is labeled
  - **Reinforcement**: No clear labels, but feedback (in the form of a reward) is accessible

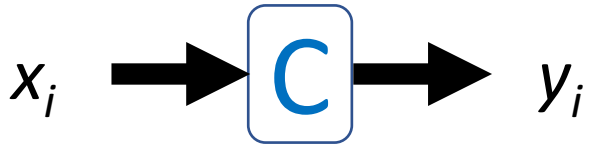
Supervised

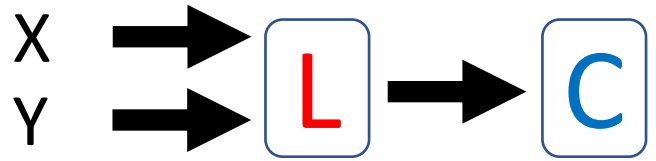
Our ~~Previous Rule-Based~~ Classifier



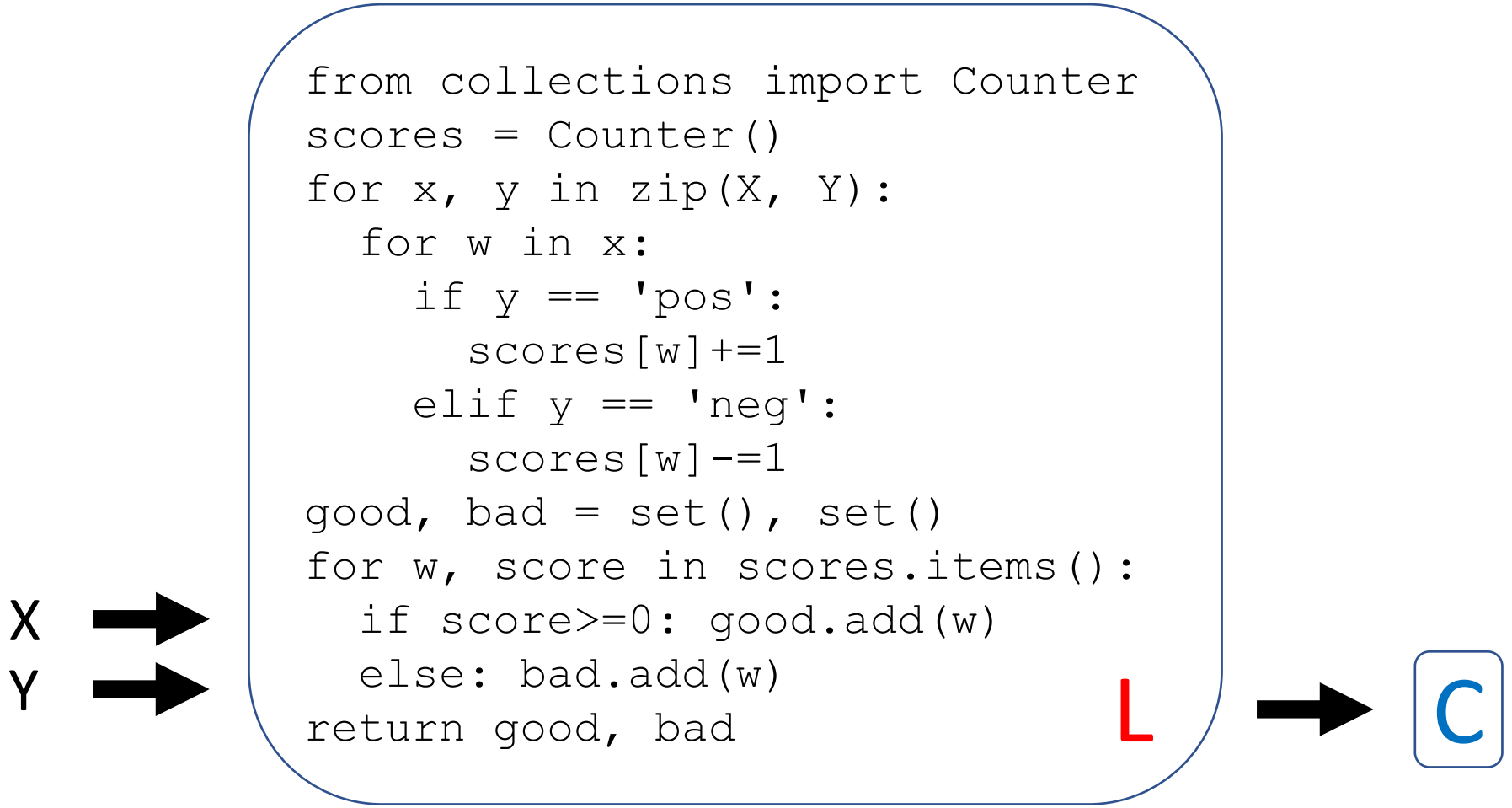
Notation

- Training examples: $X = (x_1, x_2, \ldots, x_N)$
- Labels of training examples: $Y = (y_1, y_1, \ldots, y_N)$
- A classifier C maps x_i to y_i
- A learner L infers C from (X, Y)





Counting-based Learner



A Probabilistic Classifier

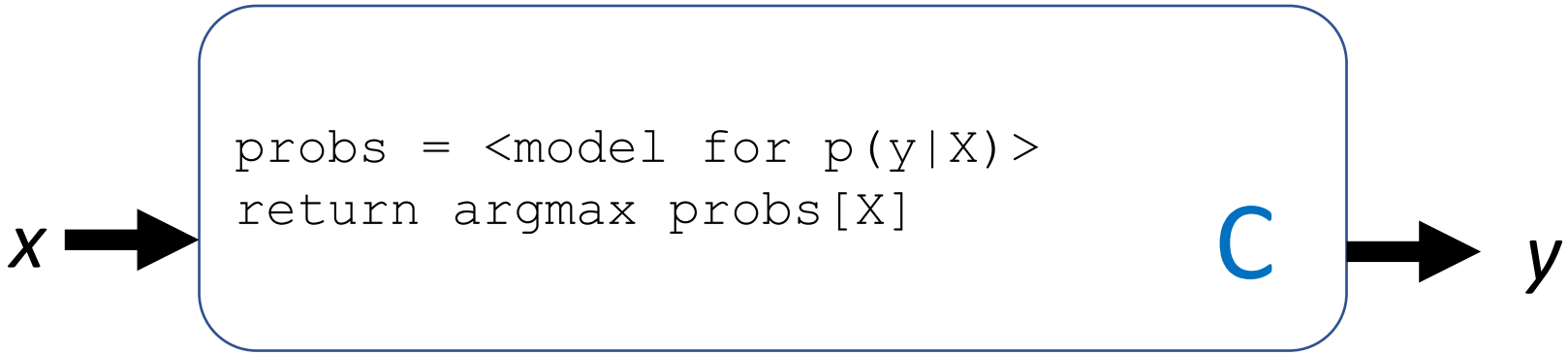
Review #34925: "Filled with horrific dialogue, laughable characters, a laughable plot, and really no interesting stakes during this film, "Star Wars Episode I: The Phantom Menace" is not at all what I wanted from a film that is supposed to be the huge opening to the segue into the fantastic Original Trilogy. The positives include the score, the sound..."

Experiment: "people wrote reviews of m

Outcomes (Ω): all possible reviews

Events: Y=pos: all positive reviews. Y=neg = all negative reviews

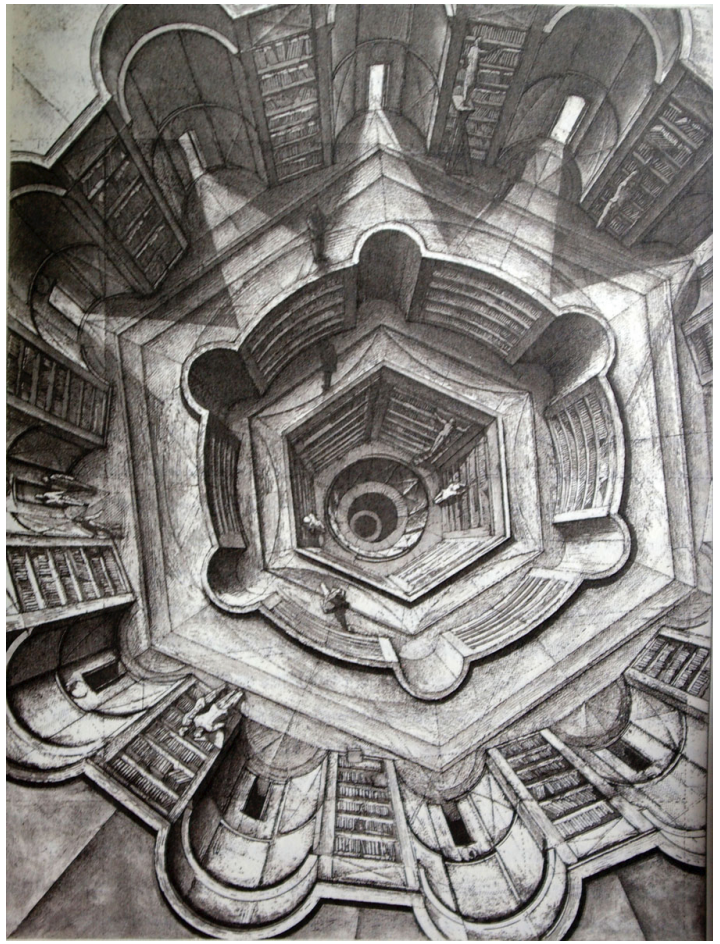
X=34925: "when review 34925 was written"



MLE is not going to work (review 34925 shouldn't be in training)!

Aside: Library of Babel (Borges, 1941)

- Contains all 1.3m-word texts, organized
- A lot of it looks like junk
- But all useful texts are in here too!
- What's it like to be a librarian?
- Look up your favorite piece of text at https://libraryofbabel.info



A probabilistic model that generalizes

- Instead of estimating p(Y|Filled with horrific...) directly, let's make two **modeling assumptions**:
  1. The **Bag of Words (BoW) assumption**: Assume the order of the words in the document doesn't matter:
     p(Y|Filled with horrific ...) = P(Y|with, horrific, Filled, ...)

a sequence

independent word events

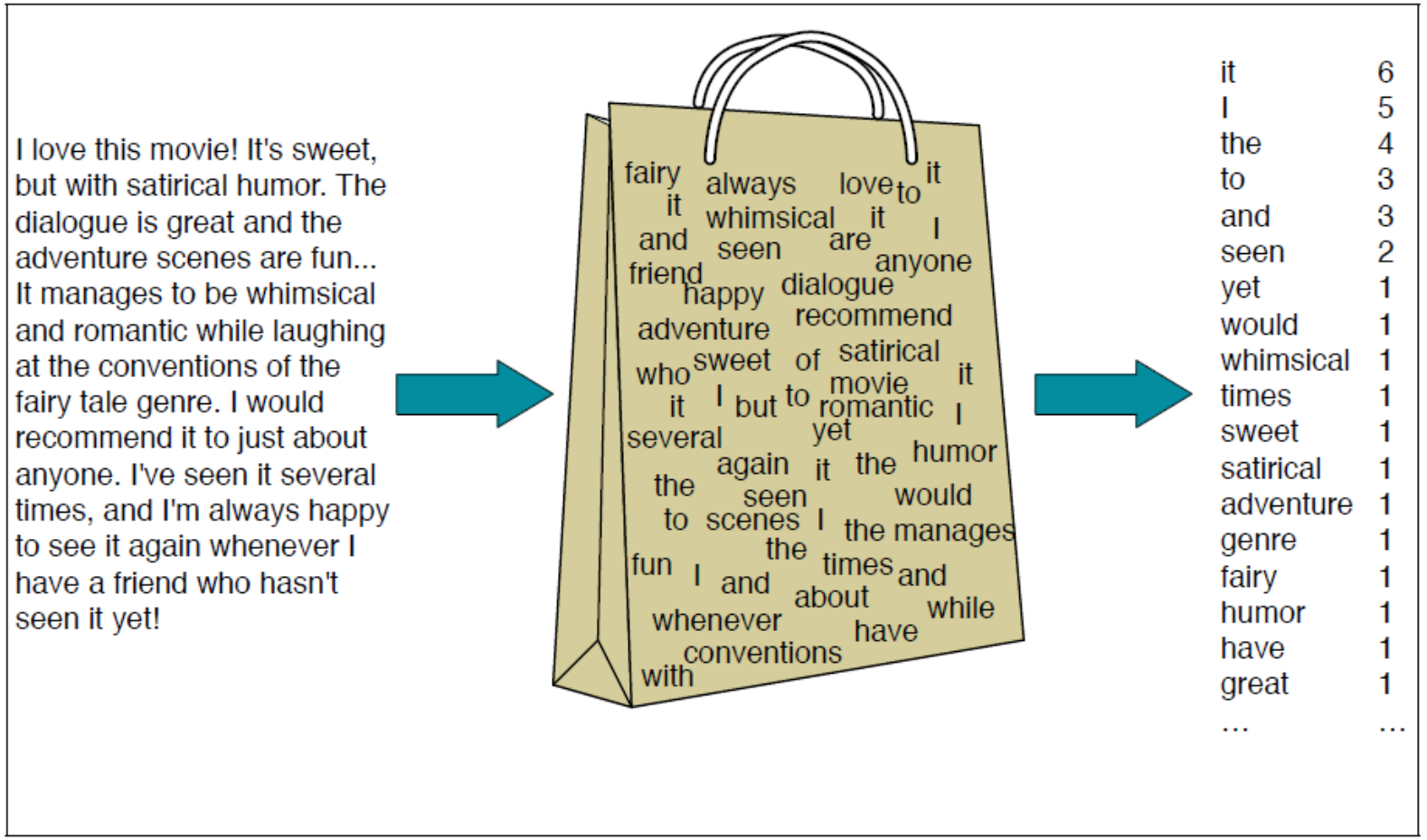


Figure 7.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Figure from J&M 3rd ed. draft, sec 6.1

A probabilistic model that generalizes

- Instead of estimating p(Y|Filled with horrific...) directly, let's make two **modeling assumptions**:
  1. The **Bag of Words (BoW) assumption**: Assume the order of the words in the document doesn't matter:
     p(Y|Filled with horrific ...) = P(Y|with, horrific, Filled, ...)

     a sequence

     independent word events

     So called because a **bag** or **multiset** is a data structure that stores counts of elements, but not their order

A probabilistic model that generalizes

- The BoW assumption isn't enough unless you happen to have seen one of 34925's anagrams in your training data (e.g. 4088293). Hence:
  2. The **naive Bayes assumption**: Words are independent conditioned on their class:
     P(Filled, with, horrific... | Y) = P(Filled | Y) x P(with | Y) x P(horrific | Y) ...
- Wait, but we were estimating P(Y|with, horrific, Filled, ...). What should we do?
- Bayes Rule!

Quiz 1

- 1) If P is a probability function, which of the following is equal to P(x | y, z)?
  - A) P(x) / P(y, z)
  - B) P(y)P(z) / P(x, y, z)
  - C) P(x, y, z) / P(y, z)
  - D) P(x)P(x|y)P(x|z)

Quiz 2

- 2) Which is/are guaranteed to be true (assume no null event cases)?
  - A) $\forall$ y $\forall$ z, Σ_x P(x | y, z) = 1
  - B) $\forall$ x, Σ_y Σ_z P(x | y, z) = 1
  - C) Σ_x P(x) = 1
  - D) $\forall$ y $\forall$ z, Σ_x P(x) P(y|x) P(z|x, y) = 1

Quiz 3

- Which of the following is/are equal to P(A|B, C, D)?
  - 1) P(B, C, D | A) P(A)
  - 2) P(B|C, D) P(B|A, C) P(B |A, D)
  - 3) P(D, C, B | A) P(A) / P(D, C, B)
  - 4) P(A | D, C, B)
- Note that while P(A|B, C, D) ≠ P(B, C, D | A) P(A), argmax_A P(A|B, C, D) = argmax_A P(B, C, D | A) P(A), and that's what we care about.

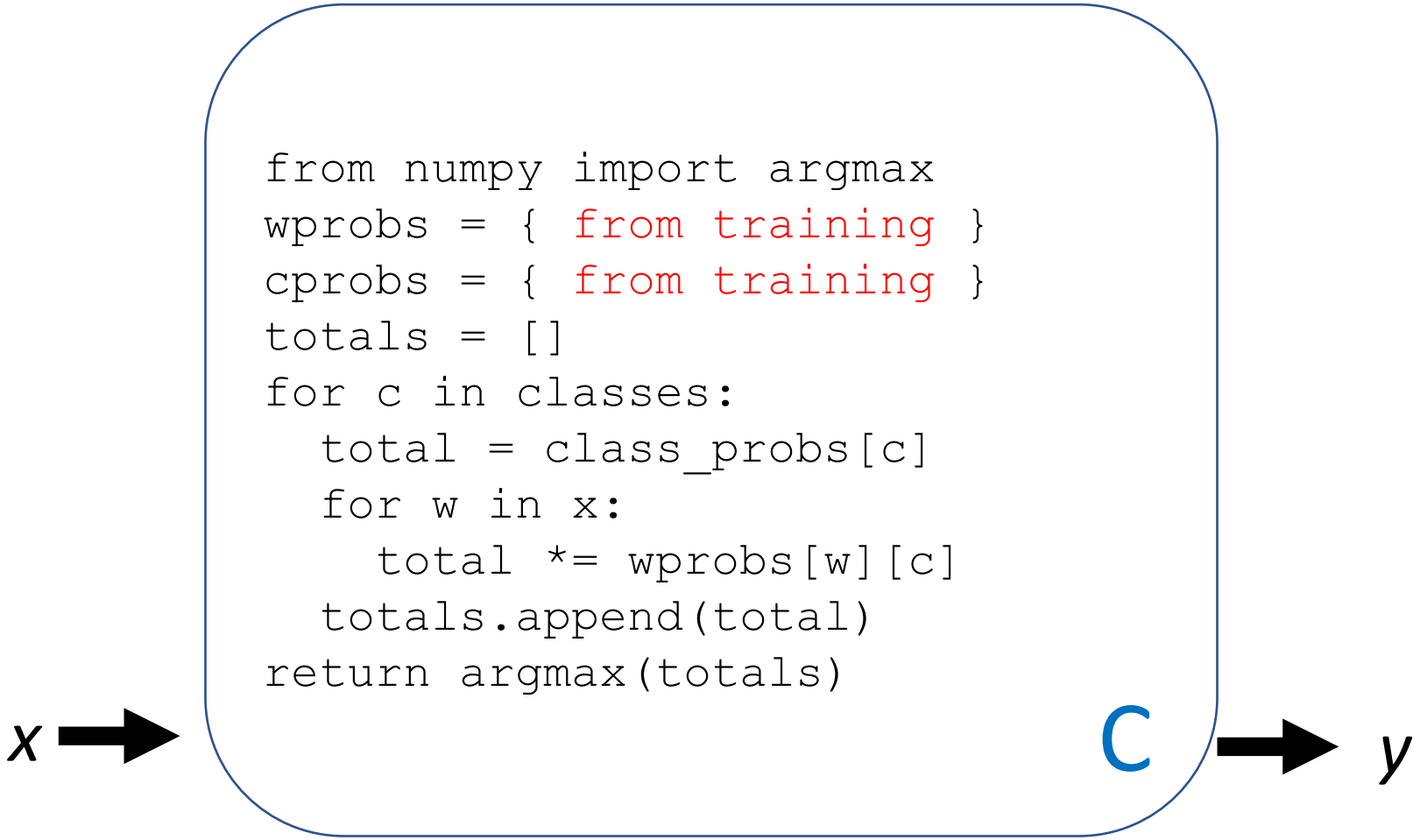
A probabilistic model that generalizes

- Put the assumptions/rule together:
  - p(Y|Filled with horrific ...) = P(Y|with, horrific, Filled, ...) (BoW assmptn)
  - $\propto$ P(Y) x P(with, horrific, Filled, ...|Y) (Bayes' rule)
  - = P(Y) x P(with|Y) x P(horrific|Y) x P(Filled|Y)... (Naive Bayes assmptn)

Is this a good model?

- "all models are wrong, but some are useful" – George Box, statistician
- What's wrong with the BoW assumption?
- What's wrong with the Naive Bayes assumption?
- But does it work?
  - Yes, for many tasks
  - And that's kind of all that matters

Naive Bayes Classifier



Naive Bayes Learner

```
from collections import Counter
cscores = Counter()
wscores = []
for c in classes:
  wscores.append(Counter())
for x, y in zip(X, Y):
  cscores[y]+=1
  for w in x.split():
    wscores[y][w]+=1
cprobs, wprobs = [], []
for c in classes:
  cprobs = cscores[c]/len(Y)
  wprob = {}
  for w, score in wscores[c].items():
    wprob[w] = score/cscores[c]
  wprobs.append(wprob)
return cprobs, wprobs
```

X →

Y →

L →

C

Parameters

- Every probability or other value that is **learned** and used by the classifier is called a **parameter** (e.g. everything the learner sent the classifier)
- Naive Bayes has two kinds of parameters:
  - Class prior distribution **P(Y)** = belief in the class
  - Likelihood distribution **P(W|Y)** = likelihood of observing a word in a class
- If there are K classes and V words, how many parameters are in a Naive Bayes classifier?

Practicalities: Smoothing

- Recall:

```
for w in x:
  total *= wprobs[w][c]
```

- What if you see a new word, or word unassociated with that class, at test time?
- Whole probability will be 0!

Laplace (add-1) smoothing

- Assume we've seen a special symbol called OOV (out of vocabulary) once per class. And assume we've seen all possibilities once more.
- Before: p(wonderful | pos) = count(wonderful, pos)/count(*, pos)
  p(terrible | pos) = count(terrible, pos) = 0/count(*, pos) = 0!

vocabulary	pos	neg
wonderful	398	17
terrible	0	228
…	…	…
TOTAL (5000 types)	147808	167585

Laplace (add-1) smoothing

- Assume we've seen a special symbol called OOV (out of vocabulary) once per class. And assume we've seen all possibilities once more.
- After: p(wonderful | pos) =
  [count(wonderful, pos)+1]/[count(*, pos) + |V|+1]

p(terrible | pos) = p(OOV | pos) = 1/[count(*, pos) + |V|+1]

vocabulary	pos	neg
wonderful	398	17
terrible	0	228
…	…	…
TOTAL (5000 types)	147808	167585

vocabulary	pos	neg
wonderful	399	18
terrible	1	229
…	…+1	…+1
OOV	1	1
TOTAL (5001 types)	147808+5001	167585+5001

Generalization: Additive (Lidstone) Smoothing

- We can use values other than 1. In general we use α
- p(wonderful | pos) =
  [count(wonderful, pos)+**α**]/[count(*, pos) + **α**|V|+**α**]

p(terrible | pos) = p(OOV | pos) = **α**/[count(*, pos) + **α**|V|+**α**]

vocabulary	pos	neg
wonderful	398	17
terrible	0	228
…	…	…
TOTAL (5000 types)	147808	167585

α=0.5

vocabulary	pos	neg
wonderful	398.5	17.5
terrible	0.5	228.5
…	…+0.5	…+0.5
OOV	0.5	0.5
TOTAL (5000 types)	147808+2500.5	167585+2500.5

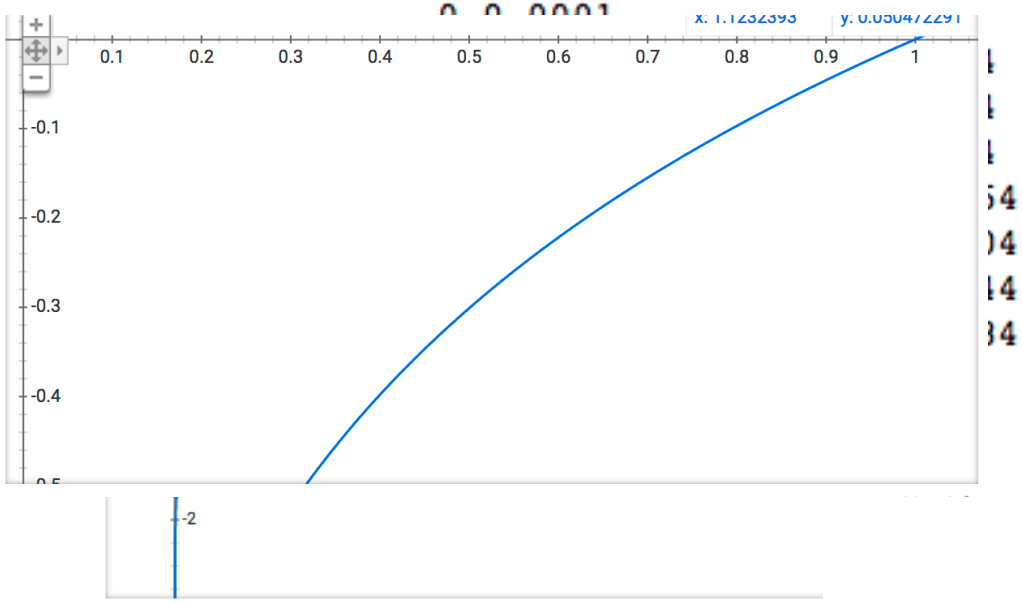
Practicalities: Underflow

- Recall:

```
for w in x:
  total *= wprobs[w][c]
```

- x may be long! wprobs[w][c] may be small!
- But remember log math!
  - exp(log(x)) = x
  - log(xy) = log(x)+log(y)
  - $\forall$ x, y > 0, x > y $\Leftrightarrow$ log(x) > log(y)

```
a=1
for i in range(100):
    a*=.0001
    if i % 10 == 0:
        print(i, a)
```



Avoiding Underflow with Logs

- Now:

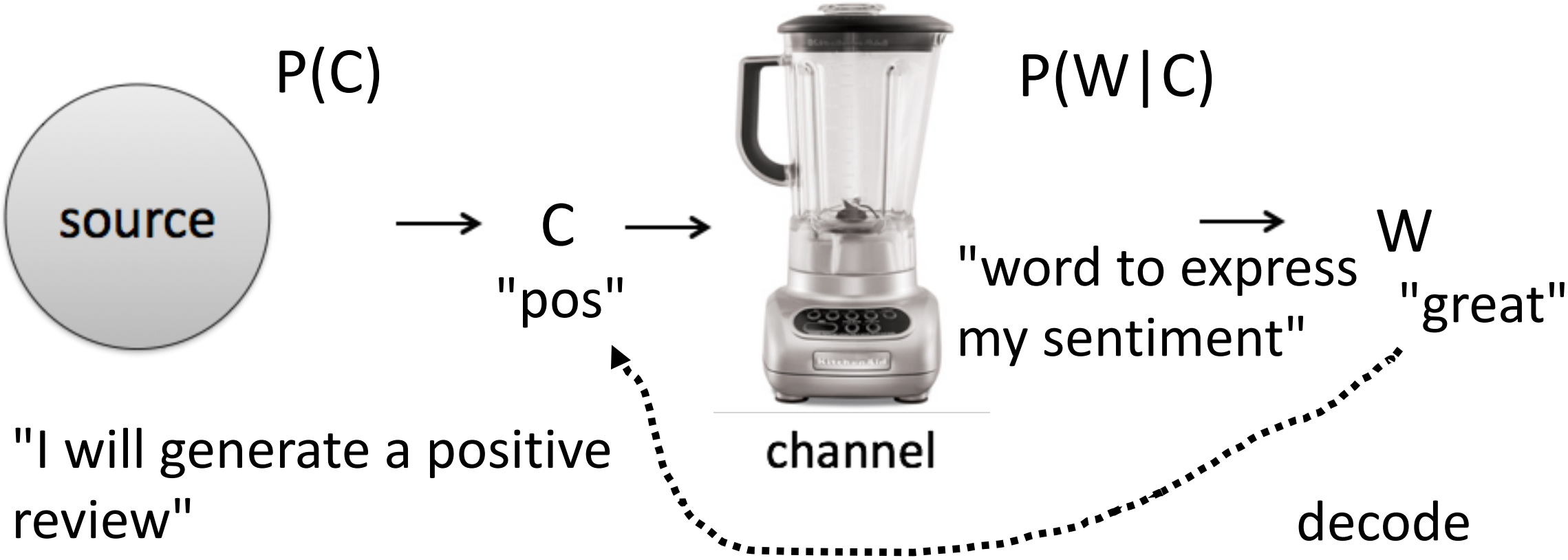
```
for w in x:
  total += log(wprobs[w][c])
```

```
from math import log
a = log(1)
for i in range(100):
    a += log(.0001)
    if i % 10 == 0:
        print(i, a)
```

```
0 -9.210340371976182
10 -101.31374409173802
20 -193.41714781149977
30 -285.5205515312616
40 -377.62395525102363
50 -469.72735897078564
60 -561.8307626905473
70 -653.9341664103088
80 -746.0375701300702
90 -838.1409738498317
```

Noisy Channel Model

- Reminder: Using Bayes' Rule interpretation of the world



Quiz 4

- Suppose we have events A, B, C, D and are given:

  $P(A) = 0.5$ $P(A,B) = 0.25$

  $P(B) = 0.3$ $P(B|C) = 0.2$

  $P(C) = 0.6$ $P(A|B,C) = 0.1$

  $P(D) = 0.4$ $P(B|A,C) = 0.7$

- Compute $P(A,B,C)$
  1. 0.3
  2. 0.012
  3. 0.5
  4. 0.15

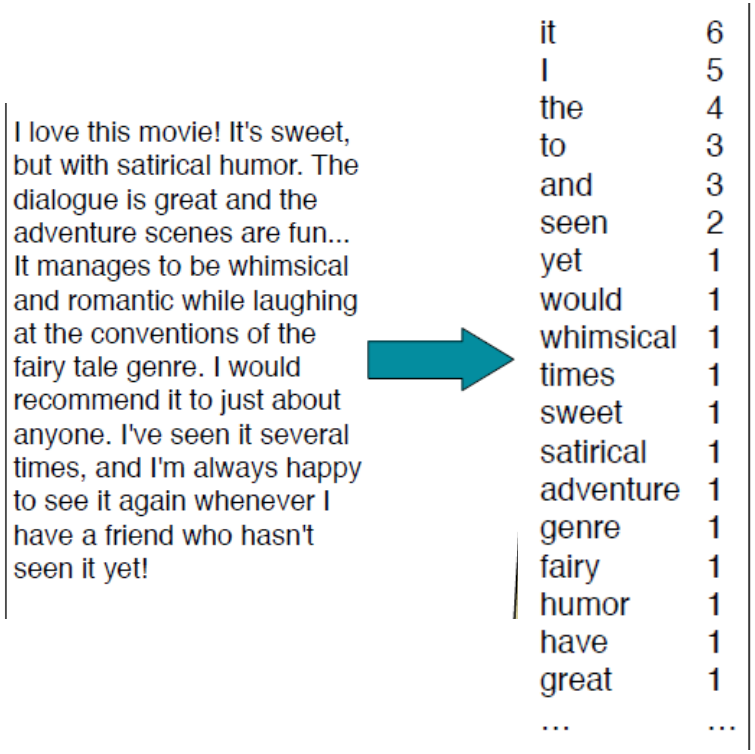
Quiz 5

	pos	neg
great	4	7
think	3	4
horrible	2	12
extra	6	8

- Consider the frequency table above. What is P(great|pos), adjusted with Laplace smoothing?
  A. 5/47
  B. 3/15
  C. 4/7
  D. 4/15
  E. 5.5/23
  F. 5/16
  G. 5/20

Feature Engineering: getting better results

- In general we call the elements of the representation we feed to a classifier/learner/predictor/etc. **features** (note: different sources may use different language here)
- So far, we have been using single words as our features (subject to a particular definition of what a word is)
- What else about a review might give us clues as to its sentiment?

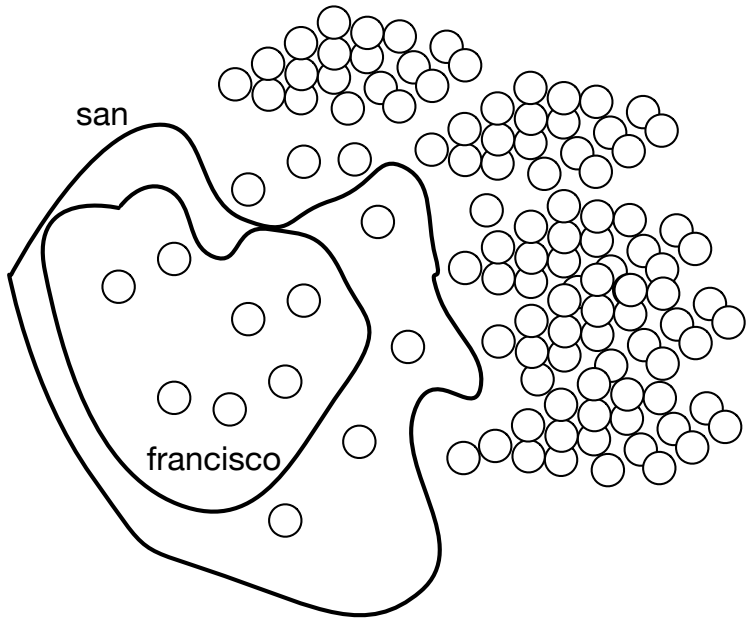


Other Features

- Bigrams ("not good", "Wall Street")
  - Aside: NLPish for lengths of word sequence: unigram, bigram, trigram, 4-gram, ...
- normalized unigrams
  - root form: "like" (instead of "liked", "liking", etc.) (cf. morphology lecture)
  - uncapitalized ("great!!!" instead of "Great!!!", "GREAT!!!")
  - punc-normalized ("Great!" instead of "Great!!!", "Great!!!!!")
  - spelling correction ("great" instead of "greate"...or "grate"??)
- length of the document (maybe positive reviews are shorter?)
  - average (max? min?) length of words
- other properties
  - contains sequence of all-caps words
  - author id
  - time of writing
- Much of this is application-dependent; try it out and see what works!

Naive Bayes Assumption Revisited

- Naive Bayes Assumption: $P(A_1, A_2, \ldots|B) = P(A_1|B)P(A_2|B)\ldots$
- Feature choice can make this assumption more naive!
  - P(san) = 11/160
  - P(francisco) = 7/160
  - P(san, francisco) = 7/160 ≈ .04
  - P(san)*P(francisco) = 77/25600 ≈ .003
- Consequence: parts of the input that are overrepresented by features have more influence
- Analogy: ask 5 people what they thought of the movie, but they all ask the same person and repeat what she said.



Feature Engineering in Labeling Tasks

- We've been making a single classification decision about a whole document but this can apply to other tasks
  - Named Entity Recognition (NER): Where are mentions of People, Organizations, Countries, etc?
  - Words Sense Disambiguation (WSD): make semantic decisions for ambiguous words (e.g. 'interest', 'bank')
  - Part of Speech (POS) Tagging: are these words nouns, adjectives, etc? (but see next lecture)
- Let's look at a WSD example:
  - x = "Wall Street vets raise concerns about interest rates , politics"
  - interest sense 1: "financial". interest sense 2: "nonfinancial"
  - We'll call the features of x $\boldsymbol{\phi}(x)$

How To Choose Features?

- Since we're using supervision (i.e. the data) to guide us we don't have to worry too much – unimportant features won't be discriminating
- But it's helpful to know something about your domain when designing features
  - are subword units important?
  - is metadata important?
  - is there something very particular about the task at hand that should be checked (e.g. detecting the writings of a psychopath who only writes in sentences with a prime number of words)
- Should I just use every possible feature I can think of?
  - More features => more flexibility
  - More features => more expensive to train
  - More features => more overfitting

What Features? What Parameters? How to Choose?

- Most of these questions are empirical: do what works best on the task you have at hand
- But beware overfitting: doing well on your training data and then doing poorly on blind test data
- Typically we divide labeled data up as follows:
  - Training data: used to build model parameters (e.g. P(good|"pos"))
  - Development/tuning data: used to test different hyperparameters/models (e.g. which α in Lidstone smoothing)
  - Test data: estimate how well you'll do on new/blind data. Don't evaluate on this until you're done/nearly done with experiments!

Feature Engineering for WSD

	$\Phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

bias feature (≈ class prior): value of 1 for every x so learned weight reflects prevalence of the class

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)

Feature Engineering for WSD

	$\Phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

spelling feature

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)

Feature Engineering for WSD

	$\Phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

token positional features

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)

Feature Engineering for WSD

	$\Phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

immediately neighboring words

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)

Feature Engineering for WSD

	$\phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

unigrams

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)
- In practice: define feature **templates** , e.g. "leftWord=*" which yield many specific features

Feature Engineering for WSD

	$\Phi(x)$
bias	1
capitalized?	0
#wordsBefore	6
#wordsAfter	3
relativeOffset	0.66
leftWord=about	1
leftWord=best	0
rightWord=rates	1
rightWord=in	0
Wall	1
Street	1
vets	1
best	0
in	0
Wall Street	1
Street vets	1
vets raise	1

bigrams

x = Wall Street vets raise concerns about interest rates , politics

- Input represented as a table of features with real values (often binary)
- In practice: define feature **templates** , e.g. "leftWord=*" which yield many specific features

Feature Engineering for WSD

	$\phi(x)$	w_1	w_2
bias	1	$\log(P(c_1))$	$\log(P(c_2))$
capitalized?	0		
#wordsBefore	6		
#wordsAfter	3		
relativeOffset	0.66		
leftWord=about	1	$\log(P(\ldots\|c_1))$	
leftWord=best	0		
rightWord=rates	1		
rightWord=in	0		
Wall	1		
Street	1		
vets	1	$\log(P(\text{vets}\|c_1))$	$\log(P(\text{vets}\|c_2))$
best	0		
in	0		
Wall Street	1		
Street vets	1		
vets raise	1		

weights learned by NB

Shouldn't these be more important?

X = Wall Street vets raise concerns about interest rates , politics

Iterate over all possible features

$$\text{argmax}_c \, w_c^T \phi(x)$$

```
for c in classes:
  total = log(class_probs[c])
  for w in x:
    total += log(wprobs[w][c])
  totals.append(total)
return argmax(totals)
```

Can we be more general?

Feature Engineering for WSD

	$\phi(x)$	w_1	w_2
bias	1	3	2.4
capitalized?	0	-3.0	-6/2
#wordsBefore	6	.22	.15
#wordsAfter	3	-.01	.03
relativeOffset	0.66	1.00	.85
leftWord=about	1	0	0
leftWord=best	0	-2.0	3.0
rightWord=rates	1	5.0	-1.2
rightWord=in	0	-1	-0.5
Wall	1	.8	-.4
Street	1	2.0	.3
vets	1	…	…
best	0		
in	0		
Wall Street	1		
Street vets	1		
vets raise	1		

less important

X = Wall Street vets raise concerns about interest rates , politics

$$\text{argmax}_c \, w_c^T \phi(x)$$

Important!

Sure, let's just come up with weights from some other source!

How about just choosing whatever weights make the classifier better?

Slight Notation Correction

	$\phi(x)$	w_1	w_2
Wall	1	.8	-.4

$\text{argmax}_c \, w_c^T \phi(x)$

	$\phi(x,y)$	w
Wall ^ y=1	1	.8
Wall ^ y=2	0	-.4

$\text{argmax}_c \, w^T \phi(x,c)$

- We've been treating features as a function of just X, with a weight vector for each possible class
- Going forward we'll treat features as a function of X and Y, with a single weight vector
- The forms are equivalent but this form makes generalizing to many classes easier

Perceptron: An Error-Driven Learner/Classifier

```
import numpy as np
from classifier import feats, classify
w = np.zeros(|feats|)
for i in range(iterations):
  for t in range(datasize):
    x, y = select(X, Y)
    # run current classifier
    y' = classify(x, w)
    if y' != y:
      w += feats(x, y)-feats(x, y')
return w
```

X →
Y →

C is a subroutine of L

L → C

```
def classify(x, w):
  scores = []
  for y in labelspace:
    scores.append(np.dot(w, feats(x,y)))
  return argmax(scores)
```

What's Going On Here (I)?

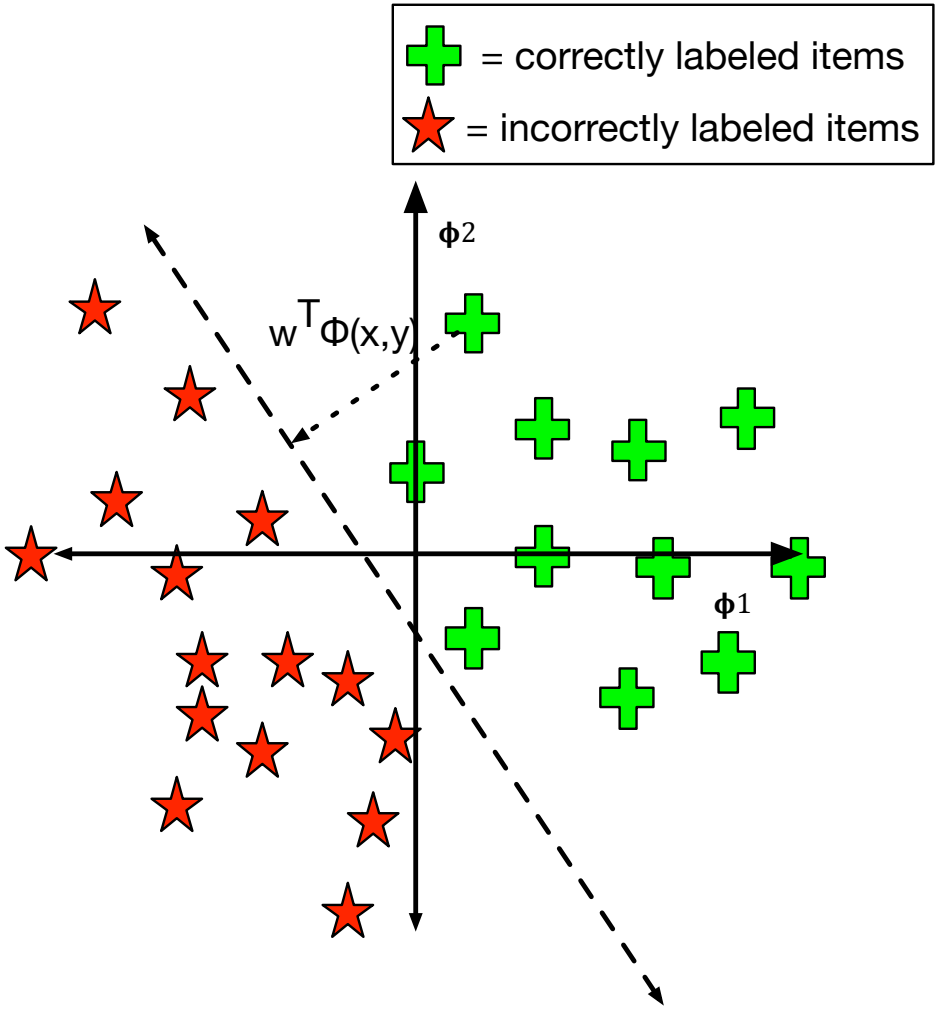
X = Wall Street vets raise concerns about interest rates , politics

```
if y' != y:
    w += feats(x, y)-feats(x, y')
```

	ϕ(x,1)	ϕ(x,2)	w	
bias ^ y=1	1	0	0.3	1.3
bias ^ y=2	0	1	0.7	-.3
Wall ^ y=1	1	0	.8	1.8
Wall ^ y=2	0	1	-.2	-1.2
this ^ y=1	0	0	2.2	2.2

- If the wrong decision was made...
- Extra weight is given to features that should have fired (but didn't)
- extra penalty is given to features that did fire (but shouldn't have).
- Features that wouldn't have fired anyway are left alone

What's Going On Here (II)?



- Each (x, y) is a point in |feature|-dimensional space.
- Weights define a hyperplane (line) along which score is zero
- Perceptron moves the line to encourage correct items to be positive distance away

What's Going On Here (III)?

- Let's say we have pair (x, y). Perceptron chooses $\text{argmax}_{y'} w^T \boldsymbol{\phi}(x,y')$; let the chosen y' be y*.
- Ideally y*=y; since we choose the argmax, we know that $w^T \boldsymbol{\phi}(x,y^*)-w^T \boldsymbol{\phi}(x,y) >= 0$. That value is called the loss; the bigger it is the worse our classifier is.
- We get to choose w. Specifically we'd like to find $\text{argmin}_{w'} w'^T \boldsymbol{\phi}(x,y^*)-w'^T \boldsymbol{\phi}(x,y)$. This is a differentiable and continuous function, so take the derivative and move away from it.

Loss Gradient for Perceptron

- $\ell_{\text{perceptron}}$ = $w^T\boldsymbol{\phi}(x,y^*)-w^T\boldsymbol{\phi}(x,y)$ unless the two are equal, then it's 0.
  - remember, w is a vector, so the gradient is taken for each member
  - however each variable appears exactly twice in the expanded equation
- $\partial/\partial w\ \ell_{\text{perceptron}} = \boldsymbol{\phi}(x,y^*)-\boldsymbol{\phi}(x,y)$ (unless the two are equal...)
- And we want a negative step, so $\boldsymbol{\phi}(x,y)-\boldsymbol{\phi}(x,y^*)$ (unless the two are equal)

Other Perceptron Notes

- Since it's not closed-form like NB, weights need to be iteratively updated for some length of time. How long? Black art.
- What order to process the data? Probably randomly shuffle, but experimentally determined
- Avoiding overfitting
  - Averaging: Keep track of all weight vectors learned, then average them
  - Early stopping: Check performance on a dev set, then stop when it stops improving
- Support Vector Machine (SVM): Variant of perceptron trying to get the separating hyperplane farthest away from all points.

Perceptron Vs. Naive Bayes

Perceptron

- Discriminative model; can only classify given x
- Not probabilistic
- Iterative solution
- No independence assumption; arbitrary feats
- Best for more accuracy (esp. SVM), medium data, more feats

Naive Bayes

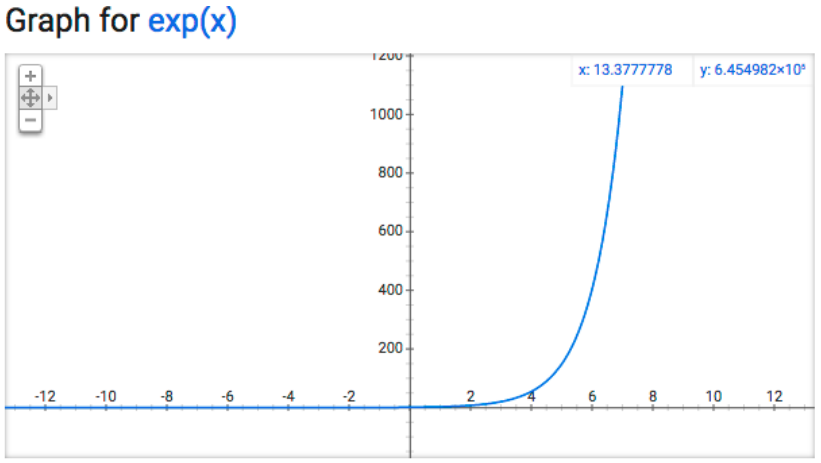
- Generative model; can be used to estimate P(x) or generate x from y
- Probabilistic
- Closed-form solution
- Overlapping features violate independence assumption
- Best for speed, lowish data, simple features

Can We Use All These Weights and Still Be Probabilistic?

- Being probabilistic allows us to quantify our uncertainty, work in pipelines, but the arbitrary weights of perceptron are freeing. Any way to have the best of both worlds?
- Recall, that the Naive Bayes objective, $P(X|Y)P(Y) = P(X, Y)$, which is also $P(Y|X)P(X)$
- In classification we're given X; we don't really need to worry about it. So if we don't want to be generative (since we're not making the Naive Bayes Assumption) we just need a good model for $P(Y|X)$
- We have a score function for the event Y, X... $w^T\boldsymbol{\phi}(x,y)$...what can we do?

Making the score function probabilistic

- $w^T \phi(x,y)$ is a score, not a probability. Further, it ranges from $-\infty$ to $+\infty$.
- Let's consider $\exp(w^T \phi(x,y))$, which is always positive, and monotone.
- We can define P(y|x) as $\frac{\exp(w^T \phi(x,y))}{\sum_{y' \in Y} \exp(w^T \phi(x,y'))}$
- Where does the denominator come from?
  - Law of total probability
- We'd like to pick ϕ to maximize this likelihood over a data set



These slides based on notes of Jacob Eisenstein

Maximizing Likelihood = Maximizing log likelihood

- We'd like to pick w to maximize $\frac{\exp(w^T\phi(x,y))}{\sum_{y'\in Y}\exp(w^T\phi(x,y'))}$ over a data set
- i.e. $\text{argmax}_w \prod_{i=1}^{n} \frac{\exp(w^T\phi(x^i,y^i))}{\sum_{y'\in Y}\exp(w^T\phi(x^i,y'))}$. Let's use our old friend log.
- = $\text{argmax}_w \sum_{i=1}^{n} \log \frac{\exp(w^T\phi(x^i,y^i))}{\sum_{y'\in Y}\exp(w^T\phi(x^i,y'))}$
- $= \text{argmax}_w \sum_{i=1}^{n} w^T\phi(x^i,y^i) - \sum_{y'\in Y}\exp(w^T\phi(x^i,y'))$
- A perfect classifier makes this as high as possible; the truth is very positive and everything else is very negative.

Maximizing Log Likelihood = Minimizing Log Loss

- The key general term for an item (x, y) is

$$w^T \phi(x, y) - \log \sum_{y' \in Y} \exp(w^T \phi(x, y'))$$

- A perfect classifier makes this as high as possible; the truth is very positive and everything else is very negative. The loss is the opposite;

$$-w^T \phi(x, y) + \log \sum_{y' \in Y} \exp(w^T \phi(x, y'))$$

- Let's minimize this by following the derivative, just like we did for perceptron

Loss Gradient for Logistic Regression (aka Maximum Entropy)

- $\ell_{\text{logreg}} = -w^T \phi(x, y) + \log \sum_{y' \in Y} \exp(w^T \phi(x, y'))$
- calculus facts:
  - ∂/∂x mx = mdx;
  - ∂/∂x log(x) = 1/x dx
  - ∂/∂x exp(x) = exp(x)dx
- ∂/∂w ℓ_{logreg} = -$\phi(x, y)$ + $\sum_{y' \in Y} \frac{\exp(w^T \phi(x,y'))}{\sum_{y'' \in Y} \exp(w^T \phi(x,y''))} \phi(x, y')$
- = -$\phi(x, y)$ + $\sum_{y' \in Y} P(y'|x)\, \phi(x, y')$. And we want to go away from the gradient, thus $\phi(x, y)$ - $\sum_{y' \in Y} P(y'|x)\, \phi(x, y')$

Gradient Update of Logistic Regression vs Perceptron

- Perceptron: w+=$\boldsymbol{\phi}$(x,y)-$\boldsymbol{\phi}$(x,y*); considers the single wrong decision, move away by constant amount
- Logreg: w $+= \phi(x, y)$ - $\sum_{y' \in Y} P(y'|x)\, \phi(x, y')$; consider every alternative decision, move away by the model's confidence in that amount

Perceptron Vs. Logistic Regression

Perceptron

- Discriminative model
- Not probabilistic
- Iterative solution
- No independence assumption
- Easy to write
- Medium-fast to learn

Logistic Regression

- Discriminative model
- Probabilistic
- Iterative solution
- No independence assumption
- Slow to write
- Slow to learn
- Foundation of deep learning optimization

Conclusions

- We have seen how **training data** and **supervised learning** can produce a better classifier
  - **Classifier** takes an *input* (such as a text document) and predicts an *output* (such as a class label)
  - **Learner** takes *training data* and produces (statistics necessary for) the classifier

Conclusions

- Because most pieces of text are unique, it's not very practical to assume the one being classified is in the training data
  - though it is in the library of Babel!
  - We need to make **modeling assumptions** that help the learner to **generalize** to unseen inputs
- The **Naive Bayes** model and **Bag of Words** assumption are a simple, fast probabilistic approach to text classification
- The **Perceptron** allows more arbitrary features at the cost of slower learning and a requirement of more data
- **Logistic Regression** adds probabilities back into the arbitrary feature and weight capability of perceptron