Lecture 5: Naive Bayes Classification

USC VSoE CSCI 544: Applied Natural Language Processing

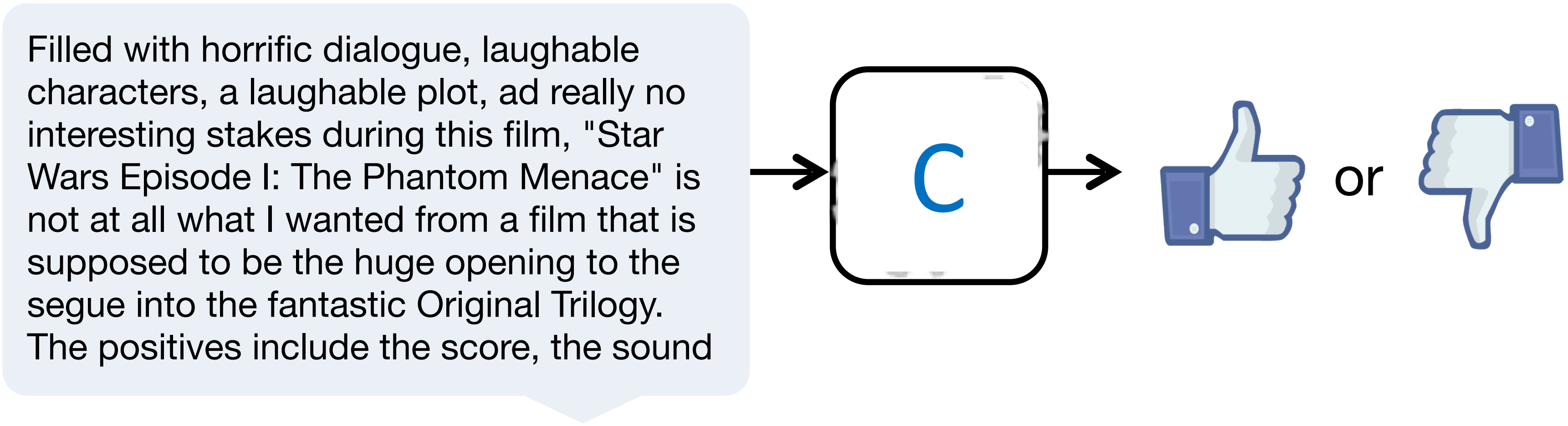
Jonathan May -- 梅約納

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 labeles
- In this lecture, input/observed data is denoted *x* and output/prediction is *y*

Our Previous Rule-Based Classifier

Note: example code from lecture 2 is slightly different but functionally equivalent

$x \rightarrow$

```
good = { 'yay', 'love', ...}
bad = { 'terrible', 'boo', ...}

score = 0
for w in x:
  if w in good:
    score +=1
  elif w in bad:
    score -=1
if score >= 0:
  return 'pos'
return 'neg'
```

C

$\rightarrow y$

from our pos/neg word list!

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

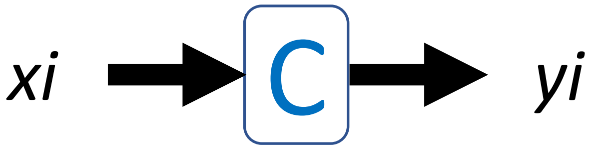
Supervised
Our ~~Previous Rule-Based~~ Classifier

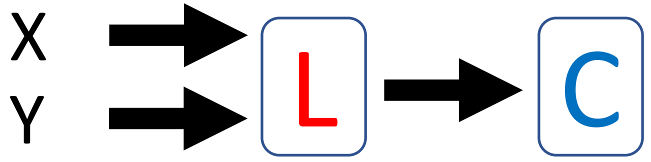
```
good = { ... from training ...}
bad = { ... from training ...}

score = 0
for w in x:
  if w in good:
    score +=1
  elif w in bad:
    score -=1
if score >= 0:
  return 'pos'
return 'neg'
```

x → C → y

Notation

- Training examples: $X = (x1, x2, ..., xN)$
- Labels of training examples: $Y = (y1, y1, ..., yN)$
- A classifier C maps *xi* to *yi*
- A learner L infers C from (*X*, *Y*)





Counting-based Learner

X

Y

```
from collections import Counter
scores = Counter()
for x, y in zip(X, Y):
  for w in x:
    if y == 'pos':
      scores[w]+=1
    elif y == 'neg':
      scores[w]-=1
good, bad = set(), set()
for w, score in scores.items():
  if score>=0: good.add(w)
  else: bad.add(w)
return good, bad
```

L

C

Probability Review Questions

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

Probability Review Questions

- 2) Which is/are guaranteed to be true?
  - A) $\forall$ y $\forall$ z, Σ_x $p(x \mid y, z) = 1$
  - B) $\forall$ x, $\Sigma_y \Sigma_z$ $p(x \mid y, z) = 1$
  - C) Σ_x $p(x) = 1$
  - D) $\forall$ y $\forall$ z, Σ_x $p(x)\, p(y|x)\, p(z|x, y) = 1$

A Probabilistic Classifier

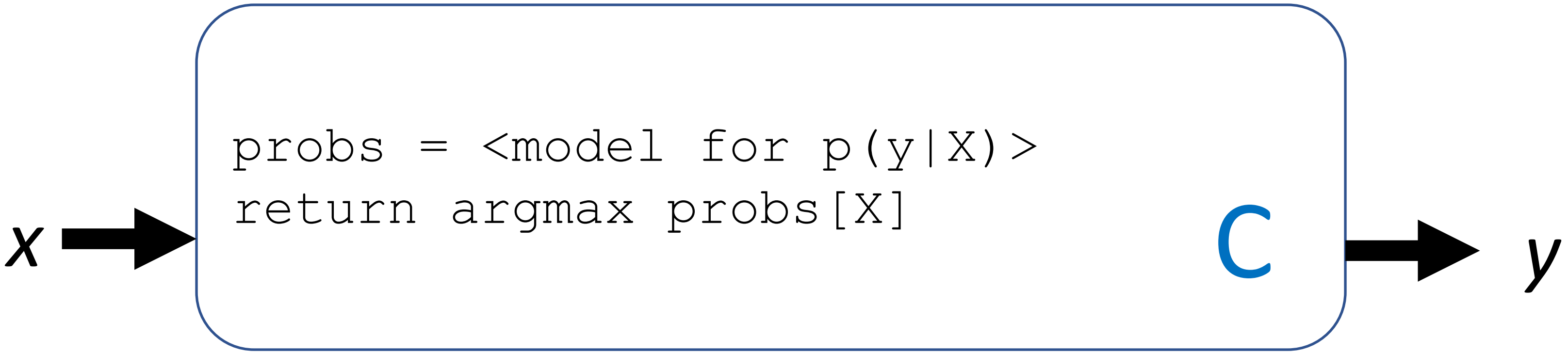
Review #34925: "Filled with horrific dialogue, laughable characters, a laughable plot, and really no interesting stakes during this film, "Star Wars Episode I: The Phantom Menace" is not at all what I wanted from a film that is supposed to be the huge opening to the segue into the fantastic Original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

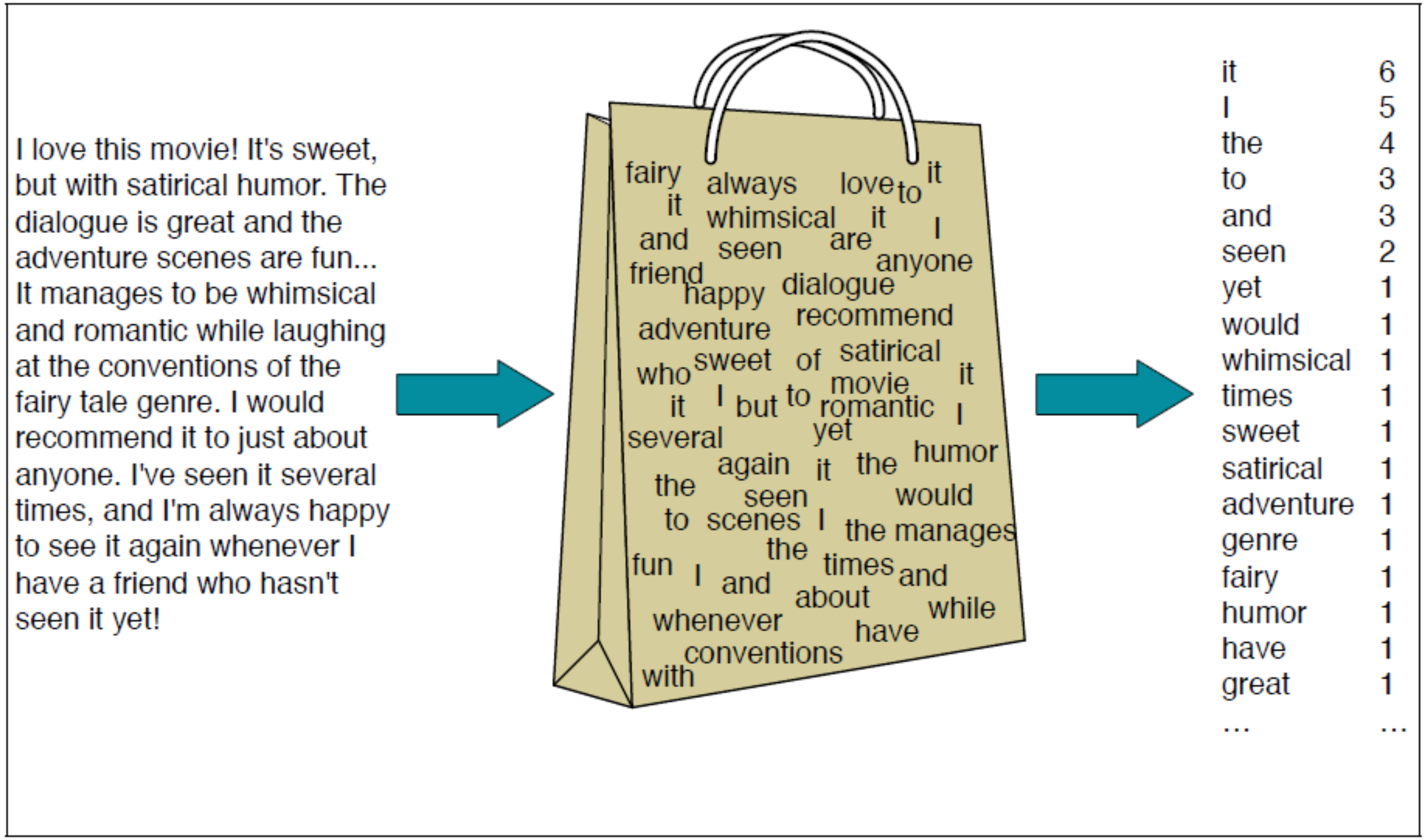


Figure 7.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Figure from J&M 3rd ed. draft, sec 7.1

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

- 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

```
from numpy import argmax
wprobs = { from training }
cprobs = { from training }
totals = []
for c in classes:
  total = class_probs[c]
  for w in x:
    total *= wprobs[w][c]
  totals.append(total)
return argmax(totals)
```

$x \rightarrow C \rightarrow y$

Naive Bayes Learner

X

Y

```
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():
    wprobs[w] = score/cscores[c]
  wprobs.append(wprob)
return cprobs, wprobs
```

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 (5000 types)	147808+5001	167585+5001

Practicalities: Underflow

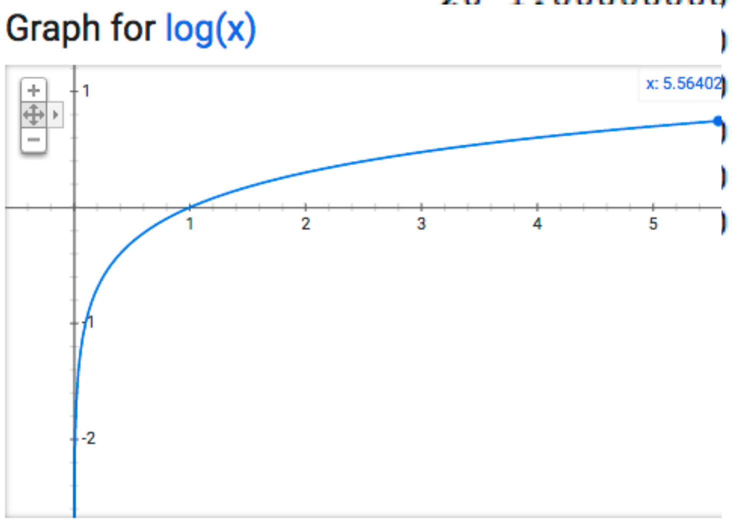
- Recall:

```
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)
```

```
0 0.0001
10 1.0000000000000003e-44
20 1.0000000000000007e-84
)0000001e-124
)00000015e-164
)00000021e-204
)00000029e-244
)00000036e-284
```



Avoiding Underflow with Logs

- Now:

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

```python
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

P(C)

P(W|C)

C

"pos"

"word to express my sentiment"

W

"great"

"I will generate a positive review"

decode

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