

# Classification: Naïve Bayes

Nathan Schneider

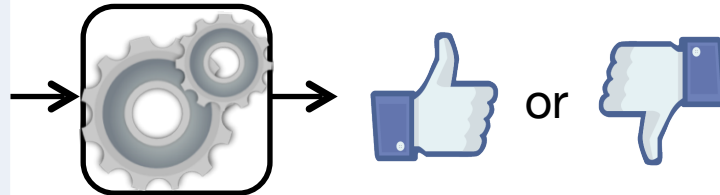
(slides adapted from Chris Dyer, Noah Smith, Sharon  
Goldwater, et al.)

ENLP | 28 January 2025

# Sentiment Analysis

- Recall the task:

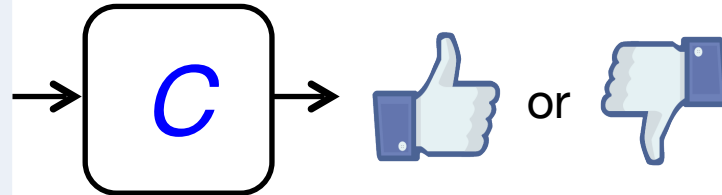
Filled with horrific dialogue, laughable characters, a laughable plot, and really no interesting stakes during this film, "Star Wars Episode I: The Phantom Menace" is not at all what I wanted from a film that is supposed to be the huge opening to the segue into the fantastic Original Trilogy. The positives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



# Sentiment Analysis

- Recall the task:

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



- This is a **classification** task: we have open-ended text as *input* and a fixed set of discrete classes as *output*.
- By convention, the input/observed information is denoted  $x$ , and the output/predicted information is  $y$ .

# A Rule-based Classifier

```
good = {'yay', 'cool', ...}  
bad  = {'ugh', ':(', ...}
```

```
score = 0
```

```
for w in x:
```

```
    if w in good:
```

```
        score += 1
```

```
    elif w in bad:
```

```
        score -= 1
```

```
return int(score>0)
```

$x \rightarrow$

$\rightarrow y$

*C*

# Supervised Classification

- We can probably do better with data
  - Our intuitions about word sentiment aren't perfect
- **Supervised** = generalizations are **learned** from **labeled** data
  - So, we need a **training** corpus of reviews with gold (correct) sentiment labels
  - And a learning algorithm
- This course: **inductive** learning algorithms—collect statistics from training corpus, but the resulting classifier does not rely on the training corpus itself

# A Rule-based Classifier

## Supervised

good = *{...from training data...}*

bad = *{...from training data...}*

score = 0

**for** w **in** x:

**if** w **in** good:

        score += 1

**elif** w **in** bad:

        score -= 1

**return** int(score>0)

$x \rightarrow$

$\rightarrow y$

**C**

# Notation

- Training examples:  $\mathbf{X} = (x_1, x_2, \dots, x_N)$

- Their categories:  $\mathbf{Y} = (y_1, y_2, \dots, y_N)$

- A **classifier**  $C$  seeks to map  $x_i$  to  $y_i$ :  $x \rightarrow \boxed{C} \rightarrow y$

- A **learner**  $L$  infers  $C$  from  $(\mathbf{X}, \mathbf{Y})$ :  
$$\begin{array}{ccc} \mathbf{X} & \rightarrow & \boxed{L} \\ \mathbf{Y} & \rightarrow & \end{array} \rightarrow \boxed{C}$$

# Counting as Learning

```
from collections import Counter
scores = Counter()
for x,y in zip(X,Y):
    for w in x:
        if y==THUMBS_UP:
            scores[w] += 1
        elif y==THUMBS_DOWN:
            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
```

$X \rightarrow$

$Y \rightarrow$

$\rightarrow C$

$L$



# Limitations

- Our classifier doesn't know that:
  - Some words are more strongly indicative of sentiment than others
  - The data may skew positive or negative (e.g., more or longer positive reviews than negative)
  - Infrequent words may occur only in the positive examples or only in the negative examples by accident
- Instead of raw counts, we can use a **probabilistic model**

# Review Questions: Conditional Probability

1. If  $p$  is a probability mass function, which is true by the definition of conditional probability:

$$p(x \mid y, z) =$$

a.  $p(x)/p(y, z)$

b.  $p(y)p(z)/p(x, y, z)$

c.  $p(x, y, z)/p(y, z)$

d.  $p(x)p(x \mid y)p(x \mid z)$

# Review Questions: Conditional Probability

2. Which is/are guaranteed to be true?

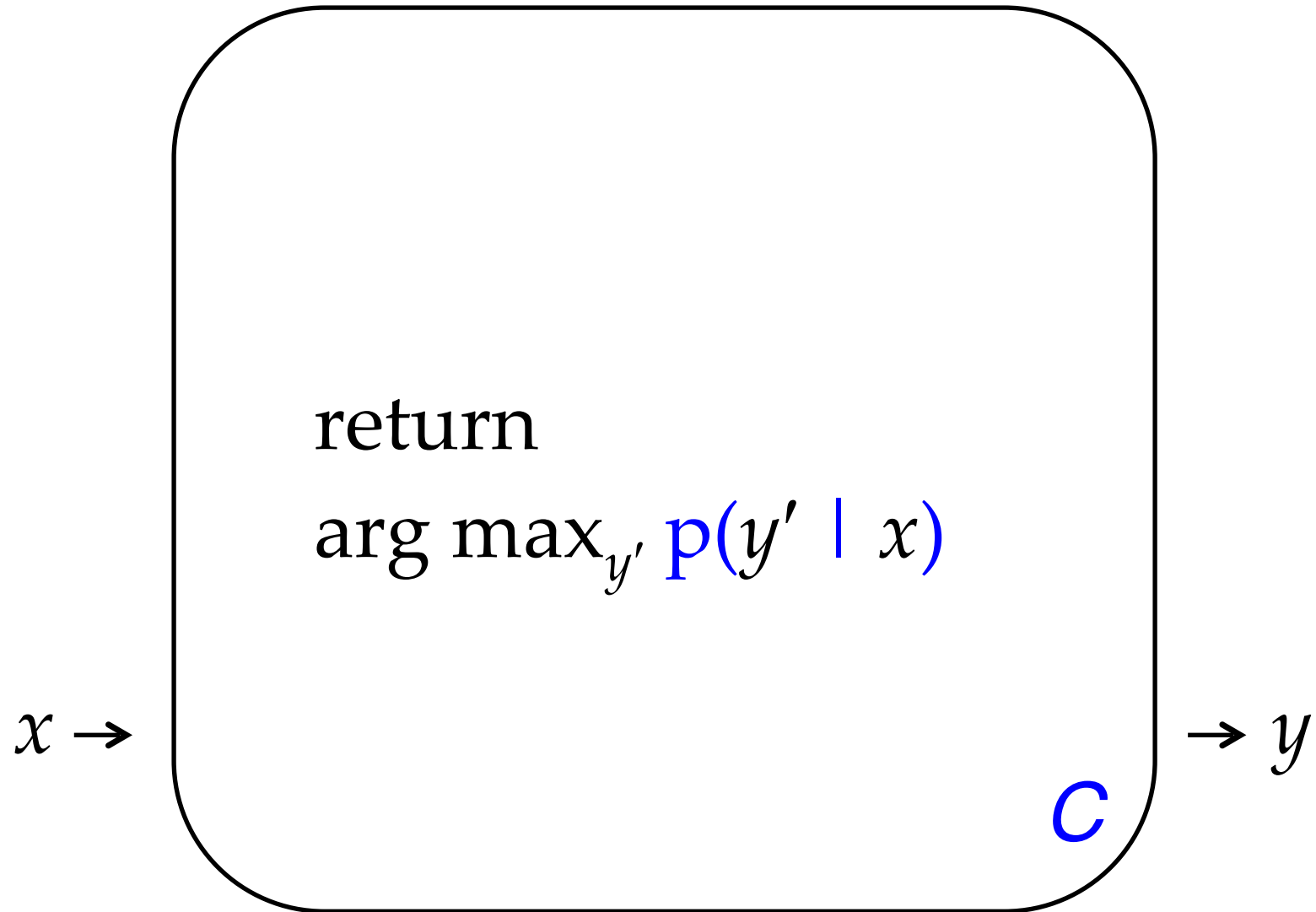
a.  $\forall y \forall z, \sum_x p(x | y, z) = 1$

b.  $\forall x, \sum_y \sum_z p(x | y, z) = 1$

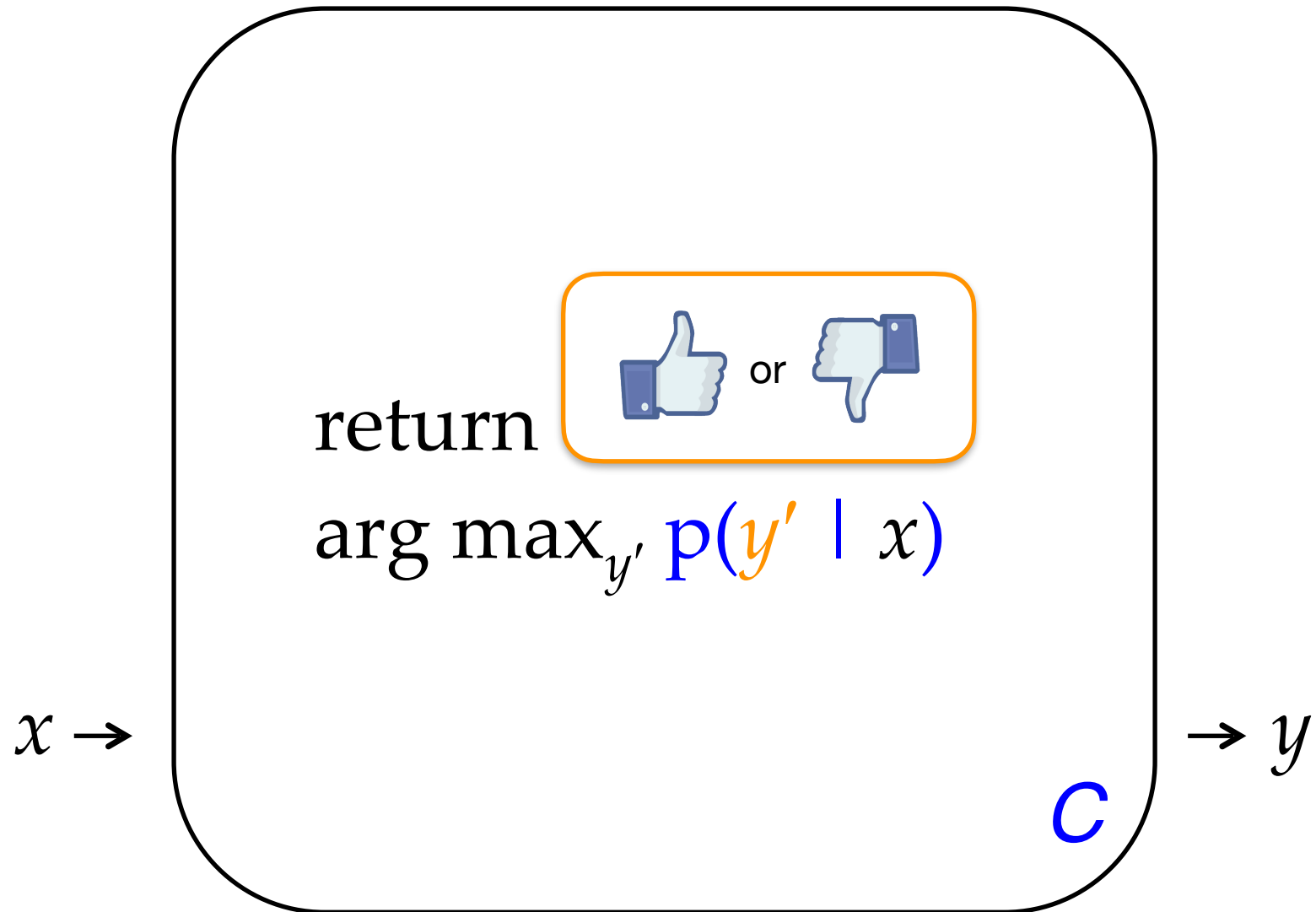
c.  $\sum_x p(x) = 1$

d.  $\forall y \forall z, \sum_x p(x)p(y|x)p(z|x,y) = 1$

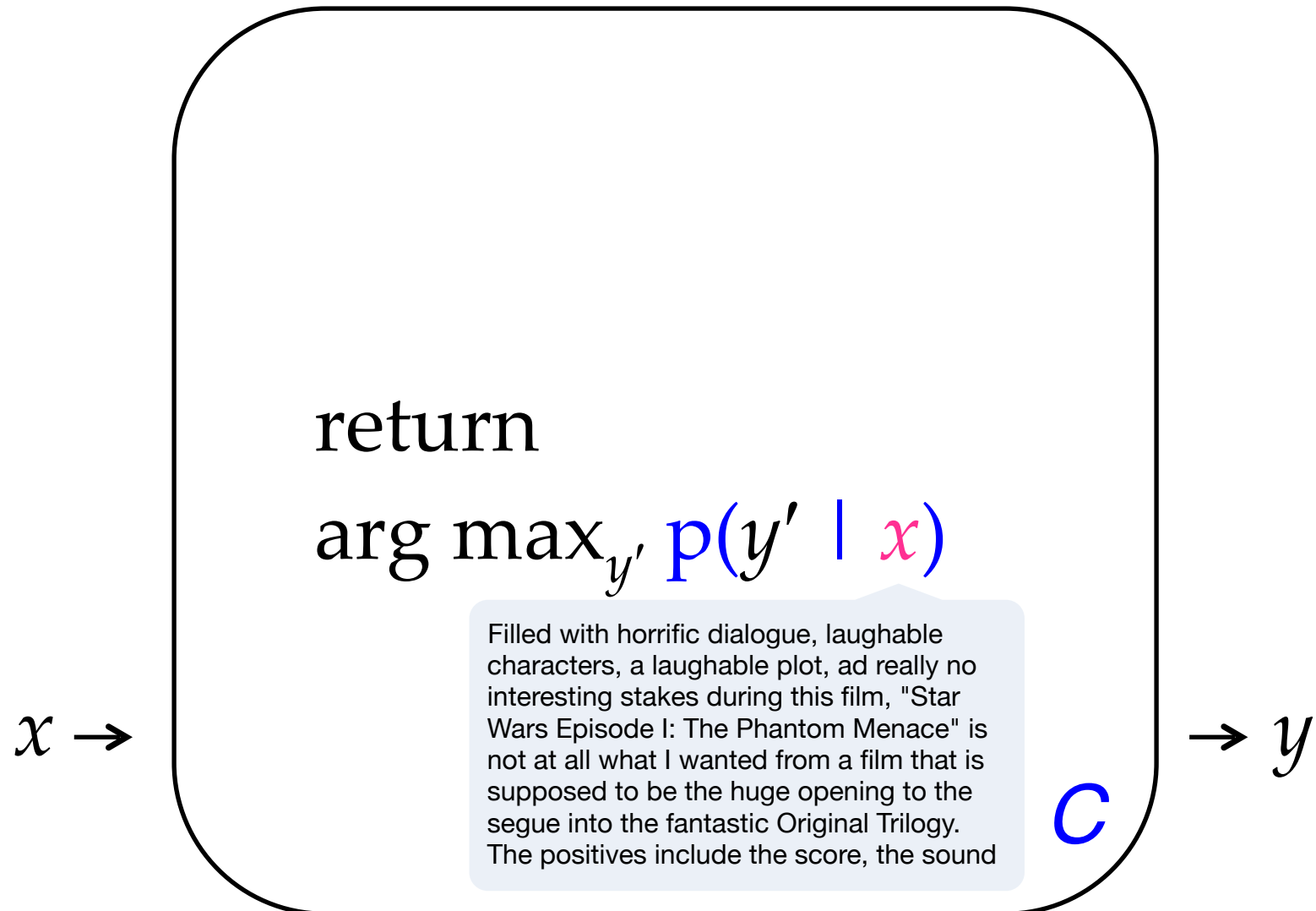
# Probabilistic Classifiers



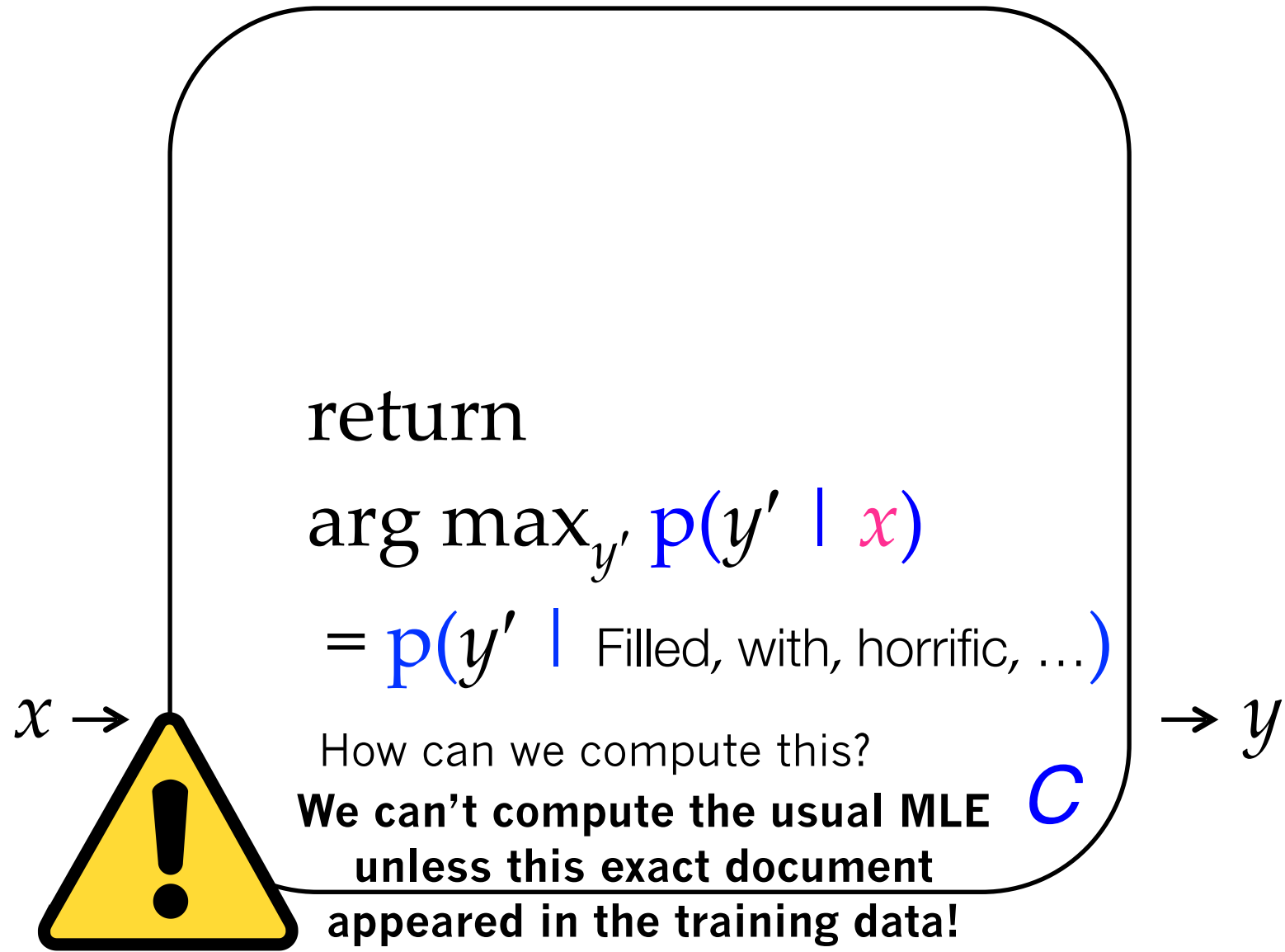
# Probabilistic Classifiers



# Probabilistic Classifiers



# Probabilistic Classifiers

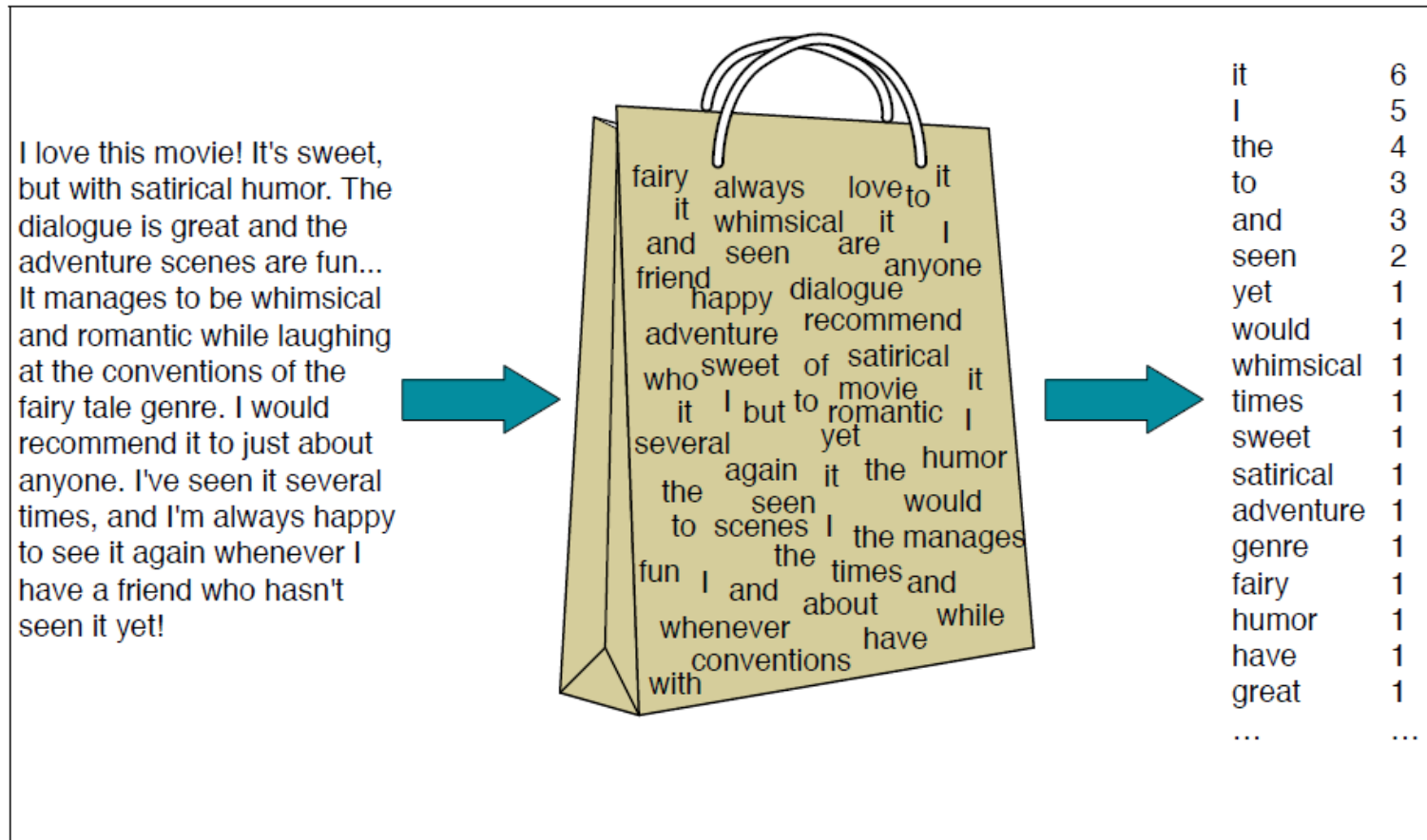


# A probabilistic model that generalizes

- Instead of estimating  $p(y' \mid \text{Filled, with, horrific, ...})$  directly, we make two **modeling assumptions**:
  1. The **Bag of Words (BoW) assumption**: Assume the order of the words in the document is irrelevant to the task. I.e., stipulate that
$$p(y' \mid \text{Filled, with, horrific}) = p(y' \mid \text{Filled, horrific, with})$$







**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  $p(y' \mid \text{Filled, with, horrific, ...})$  directly, we make two **modeling assumptions**:
  1. The **Bag of Words (BoW) assumption**: Assume the order of the words in the document is irrelevant to the task. I.e., stipulate that
$$p(y' \mid \text{Filled, with, horrific}) = p(y' \mid \text{Filled, horrific, with})$$

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, though, unless documents with all the same words occurred in the training data. Hence:

2. The **naïve Bayes assumption**: Assume the words are **independent** conditioned on the class  $y'$

$$p(\text{Filled, with, horrific} \mid y')$$

$$= p(\text{Filled} \mid y') \times p(\text{with} \mid y') \times p(\text{horrific} \mid y')$$

Hang on, we actually wanted:

$$p(y' \mid \text{Filled, with, horrific})$$

How to reverse the order?



# Bayes' Rule



$$p(B \mid A) = \frac{p(B) \times p(A \mid B)}{p(A)}$$

Prove it!



$$p(B | A) = \frac{p(B) \times p(A | B)}{p(A)}$$

multiply both sides by  $p(A)$

$$p(A) \times p(B | A) = p(B) \times p(A | B)$$

**Chain Rule**

$$p(A, B) = p(B, A)$$

...which is true by definition of joint probability

# Bayes' Rule



$$p(B | A) = \frac{p(B) \times p(A | B)}{p(A)}$$

$$\begin{array}{ccccc} p(B | A) & \propto & p(B) & \times & p(A | B) \\ \text{posterior} & & \text{prior} & & \text{likelihood} \end{array}$$



# A probabilistic model that generalizes

- The BoW assumption isn't enough, though, unless documents with all the same words occurred in the training data. Hence:

2. The **naïve Bayes assumption**: Assume the words are **independent** conditioned on the class  $y'$

$$p(\text{Filled, with, horrific} \mid y')$$

$$= p(\text{Filled} \mid y') \times p(\text{with} \mid y') \times p(\text{horrific} \mid y')$$

Hang on, we actually wanted:

$$p(y' \mid \text{Filled, with, horrific})$$

How to reverse the order?





# A probabilistic model that generalizes

- The BoW assumption isn't enough, though, unless documents with all the same words occurred in the training data. Hence:

2. The **naïve Bayes assumption**: Assume the words are **independent** conditioned on the class  $y'$

$$p(\text{Filled, with, horrific} \mid y')$$

$$= p(\text{Filled} \mid y') \times p(\text{with} \mid y') \times p(\text{horrific} \mid y')$$

$$p(y' \mid \text{Filled, with, horrific})$$

$$\propto p(y') \times p(\text{Filled, with, horrific} \mid y')$$

$$= p(y') \times p(\text{Filled} \mid y') \times p(\text{with} \mid y') \times p(\text{horrific} \mid y')$$

# Is this a good model?

- What is wrong with these assumptions?

# Is this a good model?

- George Box, statistician: “essentially, **all models are wrong, but some are useful**”)
- It turns out that naïve Bayes + BoW works pretty well for many text classification tasks, like spam detection.



# Naïve Bayes Classifier

$w_j \leftarrow [\mathbf{words}(x)]_j$

return

$\arg \max_{y'} \mathbf{p}(y') \times \prod_j \mathbf{p}(w_j \mid y')$

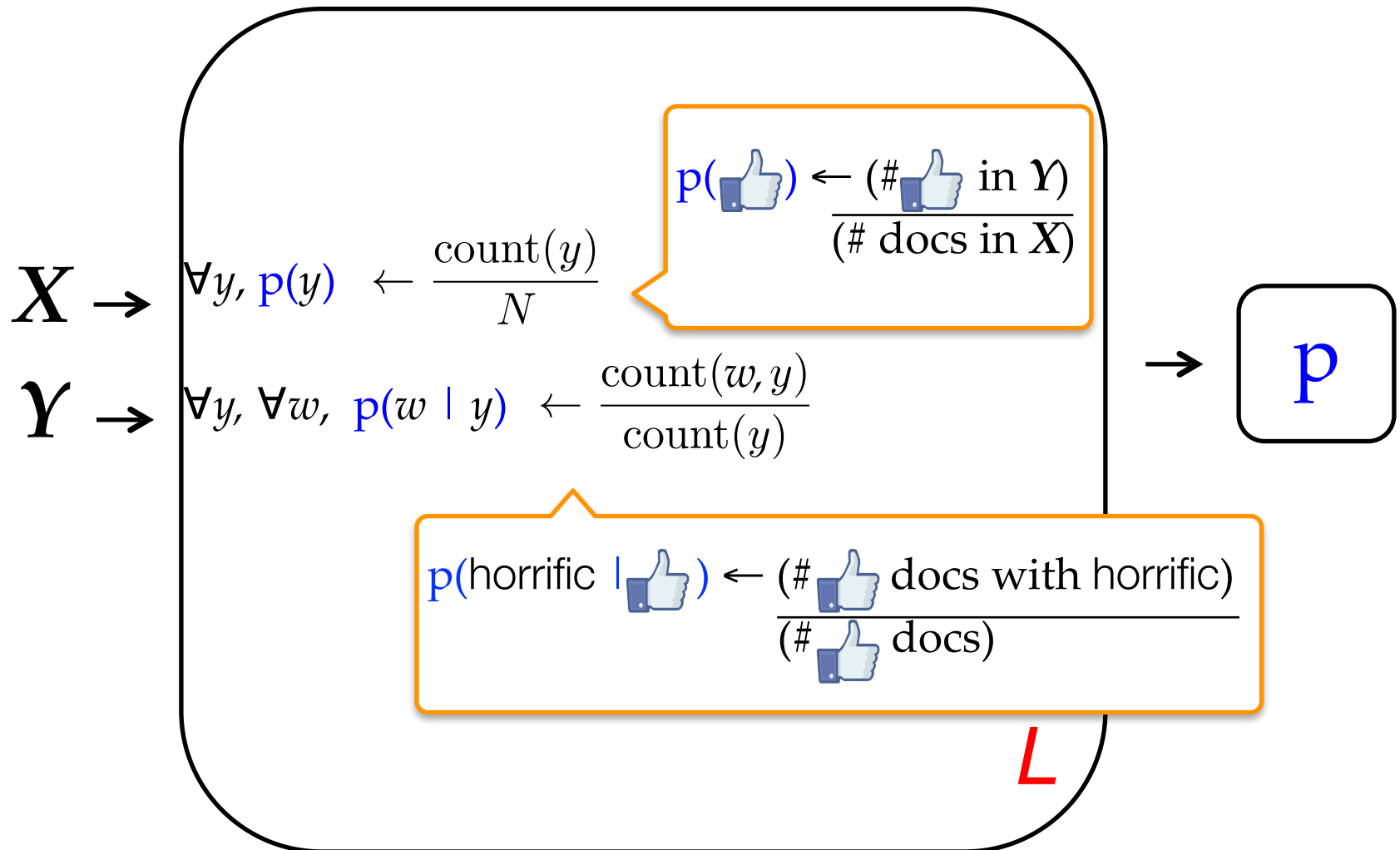
In other words: Loop over class labels,  
choose the one that makes the document  
most probable (prior  $\times$  likelihood)

**C**

$x \rightarrow$

$\rightarrow y$

# Naïve Bayes Learner



# Parameters

- Each probability (or other value) that is **learned** and used by the classifier is called a **parameter**
  - E.g., a single probability in a distribution
- Naïve Bayes has two kinds of distributions:
  - the class prior distribution,  **$p(y)$**
  - the likelihood distribution,  **$p(w \mid y)$**
- So how many parameters total, if there are  $K$  classes and  $V$  words in the training data?

# Smoothing $p(w | y)$

$$p(\text{horrific} | \text{👍}) \leftarrow \frac{(\# \text{👍 docs with horrific})}{(\# \text{👍 docs})}$$

- What if we encounter the word distraught in a test document, but it has never been seen in training?
  - Can't estimate  $p(\text{distraught} | \text{👍})$  or  $p(\text{distraught} | \text{👎})$ : numerator will be 0
  - Because the word probabilities are multiplied together for each document, the probability of the whole document will be 0



# Smoothing $p(w | y)$

$$p(\text{horrific} | \text{thumbs up}) \leftarrow \frac{(\# \text{ thumbs up docs with horrific}) + 1}{(\# \text{ thumbs up docs}) + V + 1}$$

$$p(\text{OOV} | \text{thumbs up}) \leftarrow \frac{1}{(\# \text{ thumbs up docs}) + V + 1}$$

$V$  is the size of the vocabulary of the training corpus

- **Smoothing** techniques adjust probabilities to avoid **overfitting** to the training data
  - Above: **Laplace (add-1) smoothing**
  - OOV (out-of-vocabulary/unseen) words now have small probability, which decreases the model's confidence in the prediction without ignoring the other words
  - Probability of each seen word is reduced slightly to save probability mass for unseen words



# Smoothing $p(w | y)$

$$p(\text{horrific} | \text{thumbs up}) \leftarrow \frac{(\# \text{ thumbs up docs with horrific}) + 1}{(\# \text{ thumbs up docs}) + V + 1}$$

$$p(\text{OOV} | \text{thumbs up}) \leftarrow \frac{1}{(\# \text{ thumbs up docs}) + V + 1}$$

$V$  is the size of the vocabulary of the training corpus

- **Laplace (add-1) smoothing**, above, uses a **pseudo-count** of 1, which is kind of arbitrary.
  - For some datasets, it's overkill—better to smooth less.
  - **Lidstone (add- $\alpha$ ) smoothing**: **tune** the amount of smoothing on **development** data:

$$p(\text{horrific} | \text{thumbs up}) \leftarrow \frac{(\# \text{ thumbs up docs with horrific}) + \alpha}{(\# \text{ thumbs up docs}) + \alpha(V + 1)}$$

$$p(\text{OOV} | \text{thumbs up}) \leftarrow \frac{\alpha}{(\# \text{ thumbs up docs}) + \alpha(V + 1)}$$

# Naïve Bayes Classifier

$w_j \leftarrow [\mathbf{words}(x)]_j$

return

$\arg \max_{y'} p(y') \times \prod_j p(w_j \mid y')$

In other words: Loop over class labels,  
choose the one that makes the document  
most probable (prior  $\times$  likelihood)

$C$



Can get  
very small

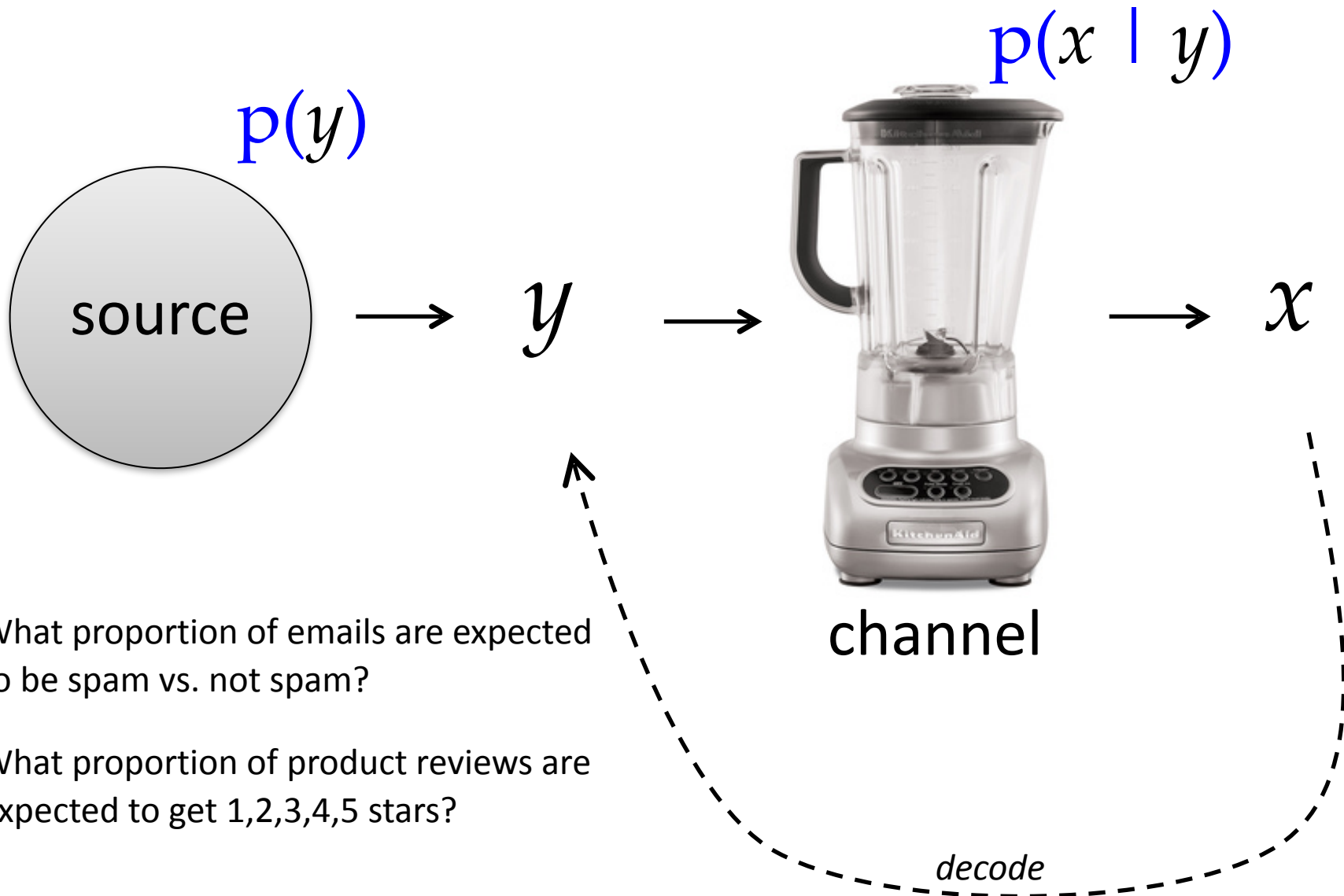
$\rightarrow y$

$x \rightarrow$

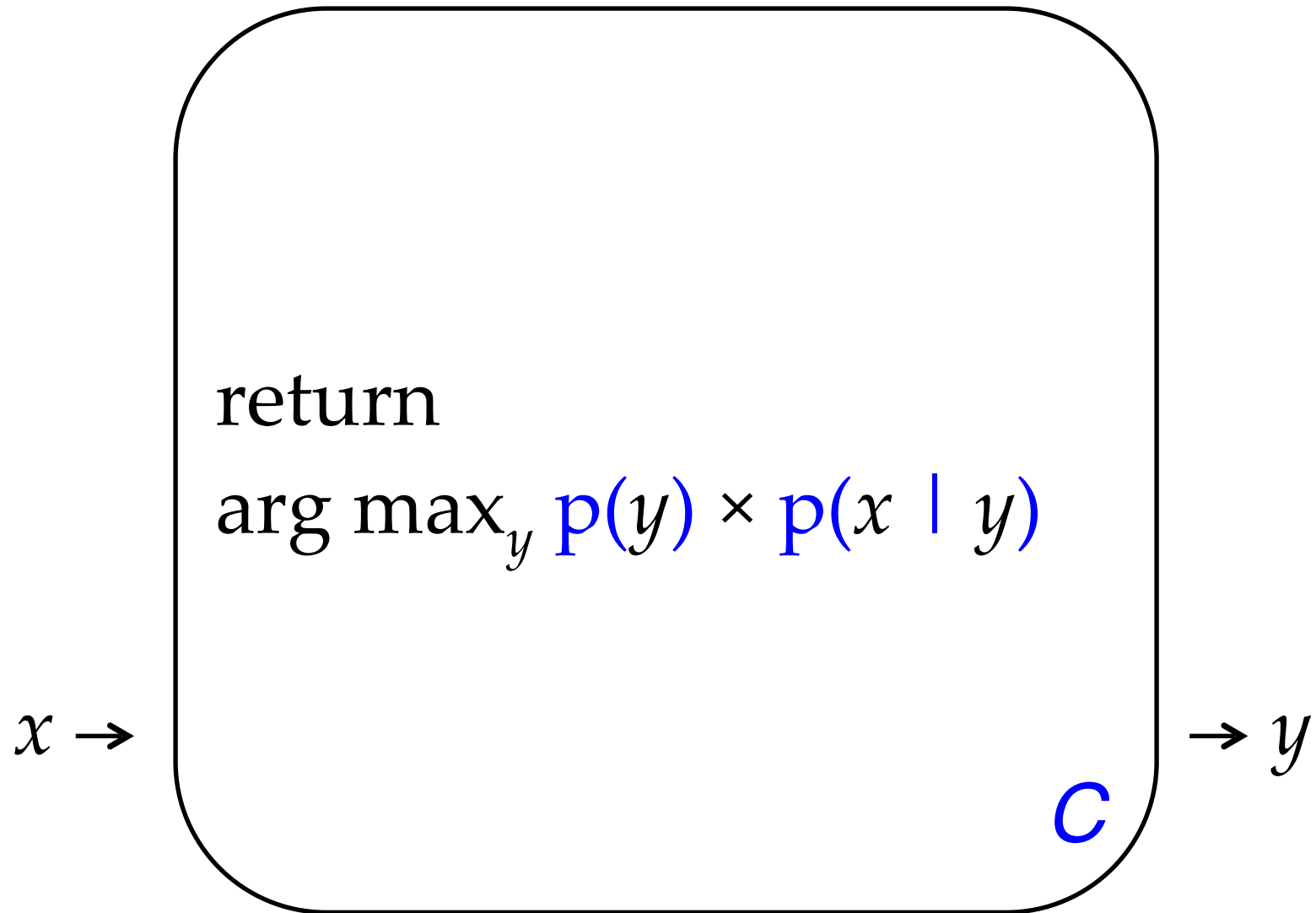
# Avoiding Underflow

- Multiplying 2 very small floating point numbers can yield a number that is too small for the computer to represent. This is called **underflow**.
- In implementing probabilistic models, we use **log probabilities** to get around this.
  - Instead of storing  $p(\bullet)$ , store  $\log p(\bullet)$
  - $p(\bullet) \times p'(\bullet) \rightarrow \log p(\bullet) + \log p'(\bullet)$
  - $p(\bullet) + p'(\bullet) \rightarrow \text{numpy.logaddexp}(\log p(\bullet), \log p'(\bullet))$

# Noisy Channel Model

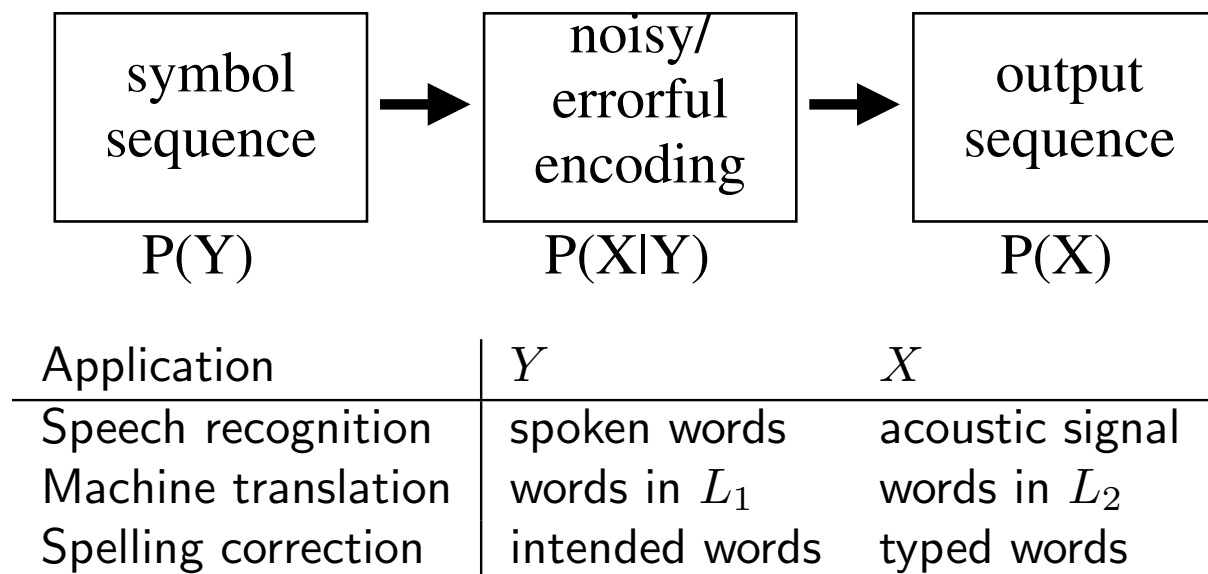


# Noisy Channel Classifiers



# Noisy Channel Model

- We imagine that someone tries to communicate a sequence to us, but noise is introduced. We only see the output sequence.



# NC Example: Spelling Correction

- $P(Y)$ : Distribution over the words the user intended to type. A language model!
- $P(X|Y)$ : Distribution describing what user is **likely** to type, given what they **meant**. Could incorporate information about common spelling errors, key positions, etc. Call it a **noise model**.
- $P(X)$ : Resulting distribution over what we actually see.
- Given some particular observation  $x$  (say, **effert**), we want to recover the most probable  $y$  that was intended.

# Conclusions

- We have seen how labeled **training data** and **supervised learning** can produce a better-informed 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 has occurred in the training data
  - We need to make **modeling assumptions** that help the learner to **generalize** to unseen inputs
- The **naïve Bayes** model + **bag-of-words** assumption are a simple, fast probabilistic approach to text classification
  - Works well for many tasks, despite being a ~~dumb~~ naïve model of language: We know that
    - \* *good, not as bad as expected  $\neq$  bad, not as good as expected*
    - \*  $p(\text{Star Wars} \mid \text{👍}) \neq p(\text{Star} \mid \text{👍}) \times p(\text{Wars} \mid \text{👍})$

# Conclusions

- In practice, we need **smoothing** to avoid assuming that everything that might come up at test time is in the training data
- Implementation trick: use **log probabilities** to avoid underflow



# Bonus: ChatGPT

# ChatGPT

- Do you think ChatGPT can help you learn **concepts** for a course?
- What problems, limitations, or risks are there with ChatGPT in general?

NA

What would be a good way to begin a second lecture in a graduate course on NLP?



A good way to begin a second lecture in a graduate course on NLP would be to review key concepts and terminology covered in the previous lecture, and then introduce new material or a specific topic of focus for the current lecture. You could also remind the students of any assignments or readings that are relevant to the current lecture. Additionally, providing an overview of the main goals or objectives of the lecture and how they align with the broader course goals can also be helpful.



A good topic to cover in a graduate course on NLP would depend on the specific focus of the course and what has been covered in previous lectures. However, some popular topics in NLP that can be covered at the graduate level include:

1. **Advanced Syntax and Grammar:** this topic could cover the theory and application of formal grammar, parsing, and syntax-based models such as Tree-adjoining Grammars and Combinatory Categorical Grammars.
2. **Semantic Representation and Understanding:** this topic could cover the study of meaning representation and natural language understanding, including distributional semantics and vector space models, and formal semantics.
3. **Machine Translation:** This topic could cover the theory and practice of machine translation, including rule-based, statistical, and neural machine translation.
4. **Dialogue Systems:** This topic could cover the design and development of natural language dialogue systems, including conversational agents and chatbots.
5. **Advanced Topics in Sentiment Analysis:** This topic could cover advanced methods and techniques for sentiment analysis and opinion mining, such as deep learning approaches and transfer learning.
6. **Advanced Topics in Text Generation:** This topic could cover advanced methods and techniques for text generation, such as neural machine translation, abstractive summarization, and text completion with GPT models.
7. **Named Entity Recognition :** This topic could cover advanced techniques for named

What are some NLP topics that would have been included in courses 10 years ago, but are no longer relevant in light of technological advances?

Looks pretty helpful, right?



In the field of Natural Language Processing, technology and research are constantly evolving, and certain topics that were once considered important may no longer be relevant in light of recent advancements. Here are a few examples of NLP topics that were likely included in courses 10 years ago but may no longer be as relevant today:

1. Rule-based Machine Translation: 10 years ago, rule-based machine translation systems were considered state-of-the-art. However, with the advancement of neural machine translation, these systems are no longer as widely used.
2. Language Modeling with n-grams: 10 years ago, n-gram-based language models were the most widely used approach for natural language processing tasks such as text generation and speech recognition. However, with the introduction of neural language models such as RNN, LSTM, and transformer-based models, n-grams are no longer as widely used.
3. Hand-crafted Features for Sentiment Analysis: 10 years ago, sentiment analysis was largely based on hand-crafted features such as lexicons and sentiment-annotated corpora. However, with the advent of deep learning techniques, this approach is no longer as popular.
4. Syntactic Parsing using CFG: 10 years ago, context-free grammars were widely used for syntactic parsing. However, with the introduction of neural models for syntactic parsing, such as the Transformer-based models, this approach is less popular.



What are some NLP topics that would have been included in courses 10 years ago, but are no longer relevant in light of technological advances?



In the field of Natural Language Processing, technology and research are constantly evolving, and certain topics that were once considered important may no longer be relevant in light of recent advancements. Here are a few examples of NLP topics that were likely included in courses 10 years ago but may no longer be as relevant today:

1. Rule-based Machine Translation: 10 years ago, rule-based machine translation systems were considered state-of-the-art. However, with the advancement of neural machine translation, these systems are no longer as widely used.
2. Language Modeling with n-grams: 10 years ago, n-gram-based language models were the most widely used approach for natural language processing tasks such as text generation and speech recognition. However, with the introduction of neural language models such as RNN, LSTM, and transformer-based models, n-grams are no longer as widely used.
3. Hand-crafted Features for Sentiment Analysis: 10 years ago, sentiment analysis was largely based on hand-crafted features such as lexicons and sentiment-annotated corpora. However, with the advent of deep learning techniques, this approach is no longer as popular.
4. Syntactic Parsing using CFG: 10 years ago, context-free grammars were widely used for syntactic parsing. However, with the introduction of neural models for syntactic parsing, such as the Transformer-based models, this approach is less popular.

X Not true!  
(Would you  
have realized  
this is false?)

What is an example of an RBMT system that was state-of-the-art 10 years ago?



RBMT stands for Rule-Based Machine Translation. One example of an RBMT system that was considered state-of-the-art around 10 years ago is the Moses machine translation system. Moses was developed by the Statistical Machine Translation group at the University of Maryland, and it was first released in 2006. It uses a combination of rule-based and statistical techniques to translate text from one language to another. Moses was known for its ability to handle multiple languages and its integration with various open-source tools such as Giza++, a word alignment tool, and SRILM, a language modeling tool. Moses was widely used by researchers and industry practitioners and it was open-source software which made it accessible to many. However, with the advent of neural machine translation, Moses and other rule-based systems like it have become less popular in recent years.

What is an example of an RBMT system that was state-of-the-art 10 years ago?



RBMT stands for Rule-Based Machine Translation. One example of an RBMT system that was considered state-of-the-art around 10 years ago is the Moses machine translation system. Moses was developed by the Statistical Machine Translation group at the University of Maryland, and it was first released in 2006. It uses a combination of rule-based and statistical techniques to translate text from one language to another. Moses was known for its ability to handle multiple languages and its integration with various open-source tools such as Giza++, a word alignment tool, and SRILM, a language modeling tool. Moses was widely used by researchers and industry practitioners and it was open-source software which made it accessible to many. However, with the advent of neural machine translation, Moses and other rule-based systems like it have become less popular in recent years.

✗ Truths mixed with falsehoods!  
Moses was a statistical, not rule-based, system. ChatGPT doesn't realize the question has an incorrect premise.