# Text Classification with Naïve Bayes

Thanks to Öznur Taştan for slides adapted from Tom Mitchell, Victor Lavrenko

# Text Classification Tasks

- Spam Detection:
  - Classify email as
    *'Spam', 'Ham'*
- Document Categorization:
  - Classify news stories into topics as
    *'Sports', 'Politics'..*
- Sentiment Analysis:
  - Classify movie reviews as
    *'favorable', 'unfavorable', 'neutral'*

# Identifying Spam

**Spam?**

*Congratulations to you as we bring to your notice, the results of the First Category draws of THE HOLLAND CASINO LOTTO PROMO INT. We are happy to inform you that you have emerged a winner under the First Category, which is part of our promotional draws.*

# What are useful features for text processing?

A **sequence of words of arbitrary length** is unsuitable for some models or computationally very expensive, and difficult to train

How to go from variable length text to a fixed size feature vector?

# How do we represent a document?

Represent the text as a **bag-of-words**

- Ignore the position of the word
- Ignore the order of the word
- Consider the words in a predefined vocabulary

Image courtesy: Joseph Gonzalez

# Bag-of-words document models

- **Document:**

  Congratulations to you as we bring to your notice, the results of the First Category draws of THE HOLLAND CASINO LOTTO PROMO INT. We are happy to inform you that you have emerged a winner under the First Category, which is part of our promotional draws.

| Term | Bernoulli | Multinomial |
|---|---|---|
| A | 1 | 1 |
| AM | 0 | 0 |
| ARE | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| CAN | 0 | 0 |
| CASINO | 1 | 1 |
| CATEGORY | 1 | 2 |
| ⋮ | ⋮ | ⋮ |
| THE | 1 | 4 |
| TO | 1 | 3 |
| WINNER | 1 | 1 |
| YOU | 1 | 3 |
| YOUR | 1 | 1 |

# Bag-of-words document models

- **Document:**

  Congratulations to you as we bring to your notice, the results of the First Category draws of THE HOLLAND CASINO LOTTO PROMO INT. We are happy to inform you that you have emerged a winner under the First Category, which is part of our promotional draws.

| Term | Bernoulli | Multinomial |
|---|---|---|
| A | 1 | 1 |
| AM | 0 | 0 |
| ARE | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| CAN | 0 | 0 |
| CASINO | 1 | 1 |
| CATEGORY | 1 | 2 |
| ⋮ | ⋮ | ⋮ |
| THE | 1 | 4 |
| TO | 1 | 3 |
| WINNER | 1 | 1 |
| YOU | 1 | 3 |
| YOUR | 1 | 1 |

**Bernoulli document model:** a document is represented by a binary feature vector, whose elements indicate absence or presence of corresponding word in the document

**Multinomial document model:** a document is represented by an integer feature vector, whose elements indicate frequency of corresponding word in the document

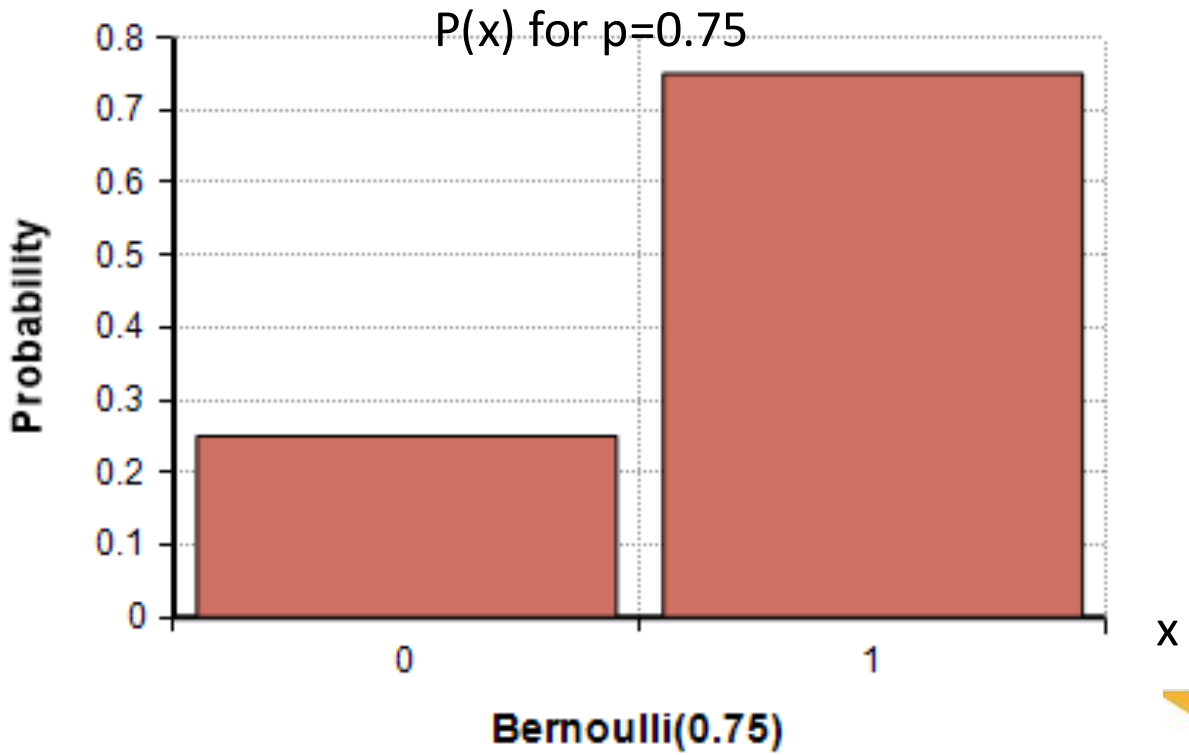
# Bernouilli Document Model

# Bernoulli Distribution

- Binary random variable x may take one of the two values:
  - success/failure, 0/1 with probabilities:

*P(x=1)= p* *Probability of success*

*P(x=0)=1-p* *Probability of failure*



# Bernoulli Distribution

- Binary random variable x may take one of the two values:
  - success/failure, 0/1 with probabilities:

$P(x=1) = p_o$ *Probability of success*
$P(x=0) = 1-p_o$ *Probability of failure*

  - **Unifying** the above, we can say: $P(x) = p_o^{\,x} (1 - p_o)^{(1-x)}$

    - Equivalently: $P(x) = x\, p_o + (1-x)(1 - p_o)$

    - The single formula helps rather than having two separate formulas (one for x=1 and one for x=0)
    - Top one is used for deriving the Maximum Likelihood estimate of $p_o$

# Bernoulli Document Model

**Features**: $X = (X_1, \ldots, X_{|V|})$ : length $|V|$ **binary vector** of word occurences

**Model's Generative Process:**

**for** each word $t$ in vocabulary:

Spin biased coin t

**if:** heads $x_t \leftarrow 1$ **else:** heads $x_t \leftarrow 0$

$x_t$=1 means word occurs in the document

| Term | Bernoulli |
|---|---|
| A | 1 |
| AM | 0 |
| ARE | 1 |
| ⋮ | ⋮ |
| CAN | 0 |
| CASINO | 1 |
| CATEGORY | 1 |
| ⋮ | ⋮ |
| THE | 1 |
| TO | 1 |
| WINNER | 1 |
| YOU | 1 |
| YOUR | 1 |

# Naïve Bayes

- Naïve Bayes assumes

$$\mathbf{P}(X_1, \ldots, X_n \mid Y) = \prod_i \mathbf{P}(X_i \mid Y)$$

- Random variables (features) $X_i$ and $X_j$ are conditionally independent of each other given the class label $Y$ for all $i \neq j$

- For training the NB classifier, we need to estimate the prior and class conditional probabilities from the data

P("money" | spam) = 0.6

P($W_{6500} = 1$ | spam) = 0.6

↳ index of word "money"

# Training a Bernoulli Model

**Training data**

Matrix $X$, document $i$ feature vector: $X_i$ presence of word $t$ in the document $i$, $X_{it}$

**Parameter Estimation(MLE)**

$$\texttt{Priors:}\mathbf{P}\left(Y = y_k\right) \approx \frac{N_k}{N}$$

$$\texttt{Likelihoods:}\mathbf{P}\left(w_t \,|\, Y = y_k\right) \approx \frac{n_k(w_t)}{N_k}$$

(Fraction of $k$ documents with word $w_t$)

spam

(# of docs. in spam category)

if k=spam

# Example

- Classify documents as `Sports` and `Informatics`

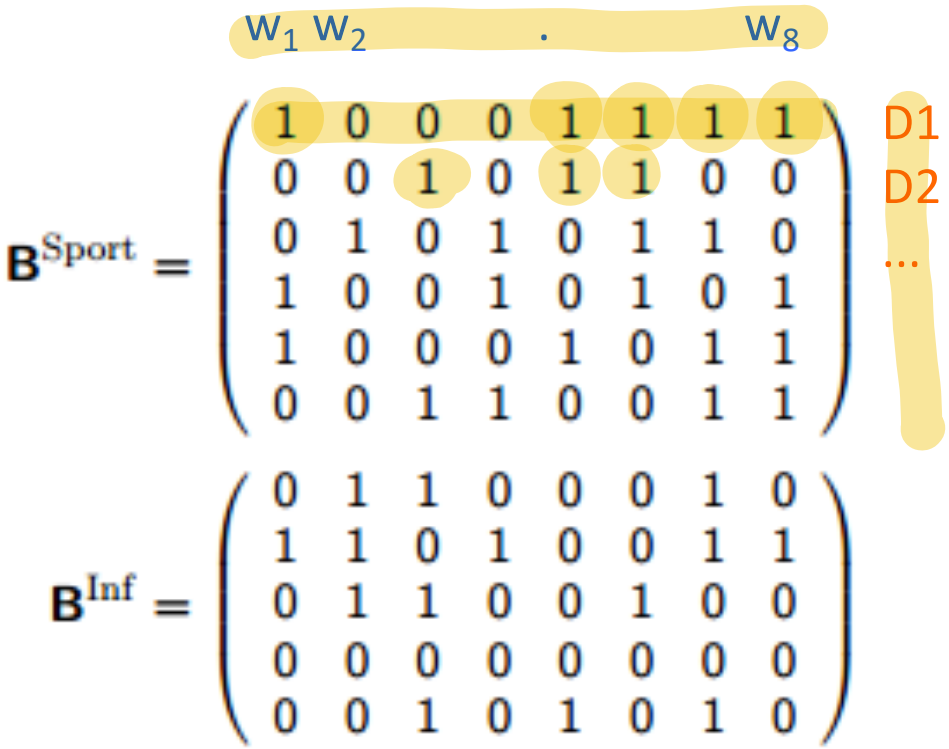
- Assume the vocabulary contains 8 words

Vocabulary contains:

$w_1$ = goal
$w_2$ = tutor
$w_3$ = variance
$w_4$ = speed
$w_5$ = drink
$w_6$ = defence
$w_7$ = performance
$w_8$ = field

$|V| = 8$

# Training Data




Vocabulary contains:

$w_1$ = goal
$w_2$ = tutor
$w_3$ = variance
$w_4$ = speed
$w_5$ = drink
$w_6$ = defence
$w_7$ = performance
$w_8$ = field

- **Rows** are **documents**
- **Columns** are **words** in the order of vocabulary

# Training Data

$$\mathbf{B}^{\text{Sport}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} \text{D1} \\ \text{D2} \\ \\ \text{...} \\ \\ \text{D6} \end{matrix}$$

$$\mathbf{B}^{\text{Inf}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Vocabulary contains:

$w_1$ = goal
$w_2$ = tutor
$w_3$ = variance
$w_4$ = speed
$w_5$ = drink
$w_6$ = defence
$w_7$ = performance
$w_8$ = field

$$\mathbf{P}(Y = Sport) = 6/11, \quad \mathbf{P}(Y = Informatics) = 5/11$$

$$(\mathbf{P}(w_t | Y = Sport)) = (3/6 \quad 1/6 \quad 2/6 \quad 3/6 \quad 3/6 \quad 4/6 \quad 4/6 \quad 4/6)$$

$$(\mathbf{P}(w_t | Y = Informatics)) = (1/5 \quad 3/5 \quad 3/5 \quad 1/5 \quad 1/5 \quad 1/5 \quad 3/5 \quad 1/5)$$

- At test time, we will get a document that will indicate whether a word in the vocabulary occurs or not in the given text.

- How do we use the Bernouilli NB model?

$$X^{new} = [1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1]$$

# Inference

- A test document:

$$X^{new} = [1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1]$$

- Priors and likelihoods:

$$\mathbf{P}(Y = Sport) = 6/11, \quad \mathbf{P}(Y = Informatics) = 5/11$$

$$(\mathbf{P}(w_t \mid Y = Sport)) = (3/6 \quad 1/6 \quad 2/6 \quad 3/6 \quad 3/6 \quad 4/6 \quad 4/6 \quad 4/6)$$

$$(\mathbf{P}(w_t \mid Y = Informatics)) = (1/5 \quad 3/5 \quad 3/5 \quad 1/5 \quad 1/5 \quad 1/5 \quad 3/5 \quad 1/5)$$

- Posterior probabilities:

$$P(Sport \mid x^{new}) \propto P(Sport) \times P(x_1=1 \mid sport) \times P(x_2=0 \mid Sport) \times \cdots P(x_8=1 \mid Sport)$$

# Inference

- A test document:

$$X^{new} = [\underset{w_1}{1} \quad \underset{w_2}{0} \quad 0 \quad \underset{\dots}{1} \quad 1 \quad 1 \quad 0 \quad \underset{w_8}{1}]$$

Probability that w2 appears in a document from the Sport category (it was seen in 1 out of 6 documents)

- Priors and likelihoods:

$$\mathbf{P}(Y = Sport) = 6/11, \quad \mathbf{P}(Y = Informatics) = 5/11$$

$$(\mathbf{P}(w_t \mid Y = Sport)) = (3/6 \quad 1/6 \quad 2/6 \quad 3/6 \quad 3/6 \quad 4/6 \quad 4/6 \quad 4/6)$$

$$(\mathbf{P}(w_t \mid Y = Informatics)) = (1/5 \quad 3/5 \quad 3/5 \quad 1/5 \quad 1/5 \quad 1/5 \quad 3/5 \quad 1/5)$$

- Posterior probabilities:

$P(Y = S \mid X^{new}) =$ $P(w_1=1 \mid S) \times P(w_2=0 \mid S) \times P(w_3=0 \mid S) \times \dots \times P(w_8=1 \mid S) \times P(S)$

# Inference

- A test document:

$$X^{new} = [1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1]$$

$w_1$ $w_2$ ... $w_8$

- Priors and likelihoods:

$$\mathbf{P}(Y = Sport) = 6/11, \quad \mathbf{P}(Y = Informatics) = 5/11$$

$$(\mathbf{P}(w_t \mid Y = Sport)) = (3/6 \quad 1/6 \quad 2/6 \quad 3/6 \quad 3/6 \quad 4/6 \quad 4/6 \quad 4/6)$$

$$(\mathbf{P}(w_t \mid Y = Informatics)) = (1/5 \quad 3/5 \quad 3/5 \quad 1/5 \quad 1/5 \quad 1/5 \quad 3/5 \quad 1/5)$$

Probability that w2 appears in a document from the Sport category (it was seen in 1 out of 6 documents)

- Posterior probabilities:

$P(Y = S \mid X^{new}) =$ $P(w_1=1 \mid S) \times P(w_2=0 \mid S) \times P(w_3=0 \mid S) \times \ldots \times P(w_8=1 \mid S) \times P(S)$

$$P(Y = S \mid X^{new}) = 3/6 \times (1-1/6) \times (1-2/6) \times 3/6 \times 3/6. \times 4/6 \times (1-4/6) \times 4/6 = 5.6 \times 10^{-3}$$

$$P(Y = I \mid X^{new}) = 9.3 \times 10^{-6}$$

# Inference with a Bernoulli Model

**Training data**

Matrix $X$, document $i$ feature vector: $X_i$ presence of word $t$ in the document $i$, $X_{it}$

*w1 w2 w3. ..*

1 0 0 0 1 1 1 1

**Parameter Estimation(MLE)**

$$\texttt{Priors:}\mathbf{P}\left(Y = y_k\right) \approx \frac{N_k}{N}$$

$$\texttt{Likelihoods:}\mathbf{P}\left(w_t \mid Y = y_k\right) \approx \frac{n_k(w_t)}{N_k}$$ (Fraction of $k$ documents with word $w_t$)

**Classify new document $D$ with feature vector $X$:**

$$\mathbf{P}\left(Y = y_k \mid X\right) \propto \mathbf{P}\left(Y = y_k\right)\mathbf{P}\left(X \mid Y = y_k\right)$$

$$\mathbf{P}\left(X \mid Y = y_k\right) = \prod_{t=1}^{|V|}[X_t\mathbf{P}\left(w_t \mid Y = y_k\right) + (1 - X_t)(1 - \mathbf{P}\left(w_t \mid Y = y_k\right)]$$

# Multinomial Naive Bayes

# Bag-of-words document models

- **Document:**

  Congratulations to you as we bring to your notice, the results of the First Category draws of THE HOLLAND CASINO LOTTO PROMO INT. We are happy to inform you that you have emerged a winner under the First Category, which is part of our promotional draws.

| Term | Bernoulli | Multinomial |
|---|---|---|
| A | 1 | 1 |
| AM | 0 | 0 |
| ARE | 1 | 1 |
| ⋮ | ⋮ | ⋮ |
| CAN | 0 | 0 |
| CASINO | 1 | 1 |
| CATEGORY | 1 | 2 |
| ⋮ | ⋮ | ⋮ |
| THE | 1 | 4 |
| TO | 1 | 3 |
| WINNER | 1 | 1 |
| YOU | 1 | 3 |
| YOUR | 1 | 1 |

Rather than the **presence or absence of a word** in each document category, we can also consider how many times each word in the vocabulary occurs in a document.

This is often more powerful- if you have enough data.

# Multinomial Document Model

**Features**: $X = (X_1, \ldots, X_{|V|})$ : length $|V|$ **integer vector** of word occurences

**Model's Generative Process:**

$x \leftarrow$ vector of zeros

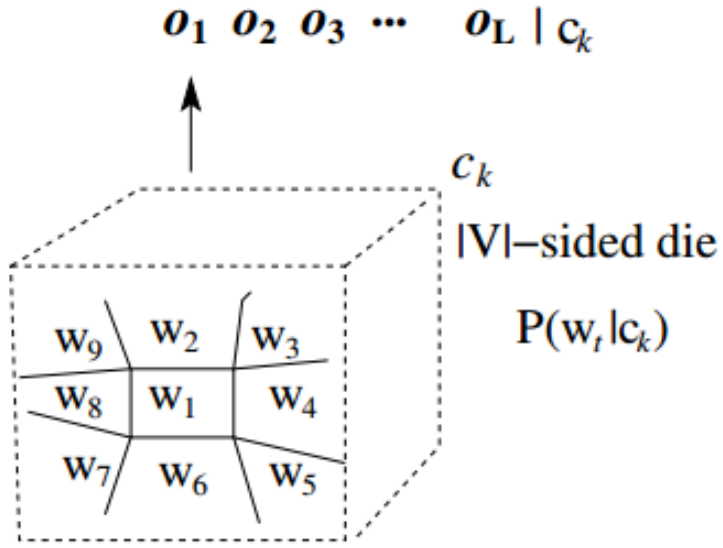
**for** each word in the document:

Roll biased $|V|$ -sided die → say $w_t$

$x_t \leftarrow x_t + 1$

Words are i.i.d. samples from a multinomial distribution.

Think like |D| |V|-sided dice throws and increasing the counts of the dice outcomes (whichever word is thrown)



# Background: Binomial Distribution

- n independent trials each of which results in success with probability of p (each a Bernouilli trial)

- Binomial distribution gives the probability distribution of the number of successes *(k out of n):*

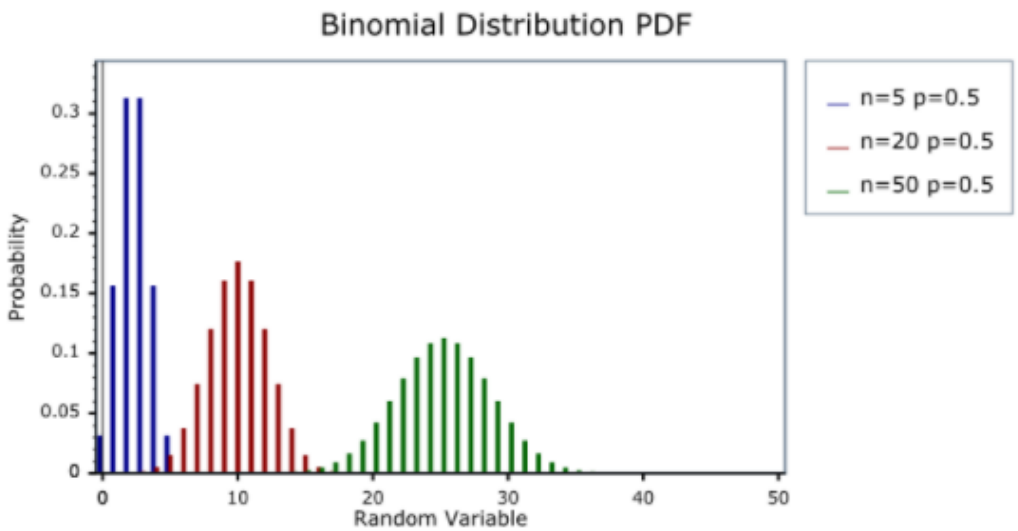
$$\Pr(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

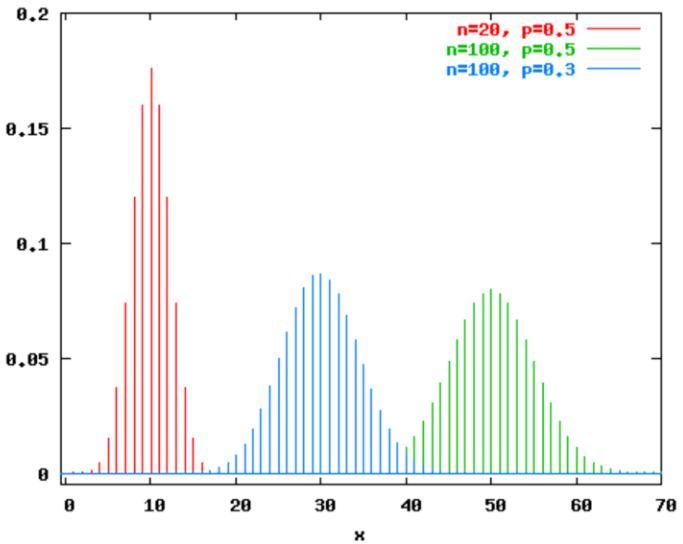
– **Example:** You flip a coin 10 times with biased probability of heads as $P_{Heads}$=0.6

– What is the probability of getting 8 H, 2T? P(k=8;n=10,p=0.6) = (10 choose 8) $p^8$x$(1\text{-}p)^2$=0.12

– What is the probability of getting 6 H, 4T? P(k=6;n=10,p=0.6) = (10 choose 6) $p^6$x$(1\text{-}p)^4$=0.25

(10 choose 8) = 45 (10 choose 6) = 210



# Multinomial Distribution

- Generalization of Binomial distribution
- **n independent trials, each of which results in one of the k outcomes with probabilities $p_k$ (example: dice).**
- Multinomial distribution gives the probability of *any particular combination of numbers of successes for the various categories k.*
  - e.g. You have a dice with k faces and corresponding probabilities $p_1 \ldots p_k$.
  - e.g. What is the probability of getting $x_1$ 1s, $x_2$ 2s and $x_3$ 3s out of n dice throws? E.g. In a10 unbiased dice throw, you get one 1, one 2, eight 6s: P(1,1,0,0,0,8) =

$P(x_1 \ldots x_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$

// Imagine for k=2

$\binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$

# General Example

We have 12 rolls and counts

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (2, 3, 0, 1, 2, 4).$$

**General (not necessarily fair) die:**

$$P(2,3,0,1,2,4) = \frac{12!}{2!\,3!\,0!\,1!\,2!\,4!}\, p_1^2\, p_2^3\, p_3^0\, p_4^1\, p_5^2\, p_6^4$$

# Multinomial Distribution

$$P(x_1 \ldots x_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

- In 12 independent dice rolls with an unbiased dice, you get nothing but 6s. What is the probability of this event?

- In 12 independent dice rolls with an unbiased dice, you get two of each dice outcome. What is the probability of this event?

# Multinomial Distribution

$$P(x_1 \ldots x_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

- In 12 independent dice rolls with an unbiased dice, you get nothing but 6s. What is the probability of this event?

$$P(0,0,0,0,0,12) = \frac{12!}{0!\,0!\,0!\,0!\,0!\,12!}\, p_1^0\, p_2^0\, p_3^0\, p_4^0\, p_5^0\, p_6^{12}$$

$$P(0,0,0,0,0,12) = 1 \cdot \left(\frac{1}{6}\right)^{12} = \frac{1}{6^{12}} = \frac{1}{2\,176\,782\,336} \approx 4.59 \times 10^{-10}$$

- In 12 independent dice rolls with an unbiased dice, you get two of each dice outcome. What is the probability of this event?

$$P(2,2,2,2,2,2) = \frac{12!}{(2!)^6} \left(\frac{1}{6}\right)^{12} = \frac{7\,484\,400}{6^{12}} = \frac{7\,484\,400}{2\,176\,782\,336} \approx 0.00344$$

# Training a Multinomial Model

**Training data:**

Matrix $X$, document $i$ feature vector: $X_i$

$X_{i,t}$: the count of number of times $w_t$ occurs in document $i$

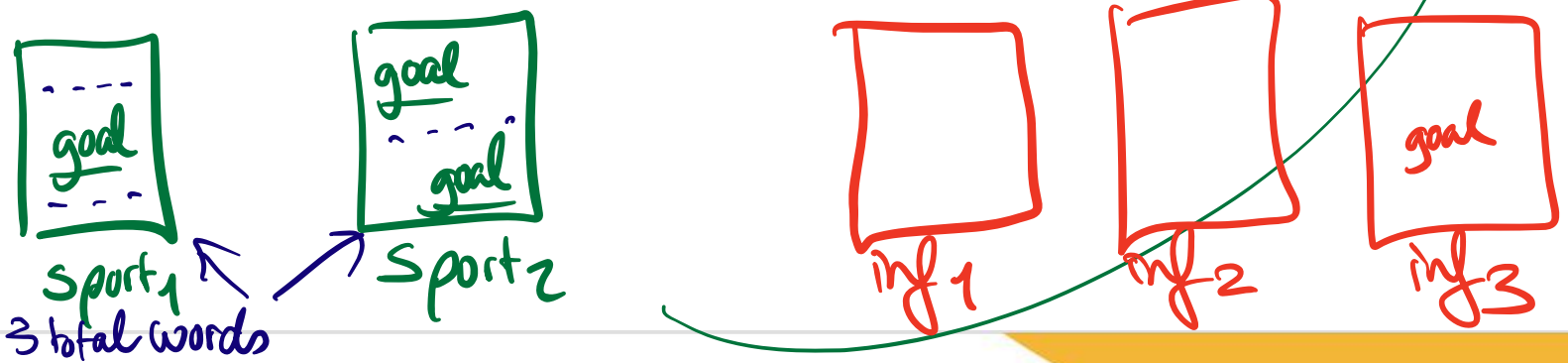
$z_{i,k}$: if document $i$ is of class $y_k$, 0 otherwise

**Parameter estimation (MLE)**

$$\texttt{Priors:}\mathbf{P}\left(Y = y_k\right) \approx \frac{N_k}{N}$$

$$\texttt{Likelihoods:}\mathbf{P}\left(w_t \mid Y = y_k\right) \approx \frac{\sum_{i=1}^{N} X_{i,t} z_{i,k}}{\sum_{t'=1}^{|V|} \sum_{i=1}^{N} X_{i,t'} z_{i,k}}$$

The relative frequency of $w_t$ in documents of class $Y = y_k$ with respect to the total number of words in documents of that class



- At inference time, we will get a document that will indicate how many times each word in the vocabulary occur.

- How do we use the multinomial NB model?

$$\mathbf{X}^{new} = \begin{bmatrix} 3 & 1 & 0 & 0 \end{bmatrix}$$

$$P(\begin{bmatrix} 3 & 1 & 0 & 0 \end{bmatrix} | S) = P(w_1 | S)^3 \times P(w_2 | S)^1 \times P(w_3 | S)^0 \times P(w_4 | S)^0$$

# Inference with a Multinomial Model

**Training data:**

Matrix $X$, document $i$ feature vector: $X_i$

$X_{i,t}$: the count of number of times $w_t$ occurs in document $i$

$z_{i,k}$: if document $i$ is of class $y_k$, 0 otherwise

**Parameter estimation (MLE)**

$$\texttt{Priors:}\mathbf{P}\left(Y = y_k\right) \approx \frac{N_k}{N}$$

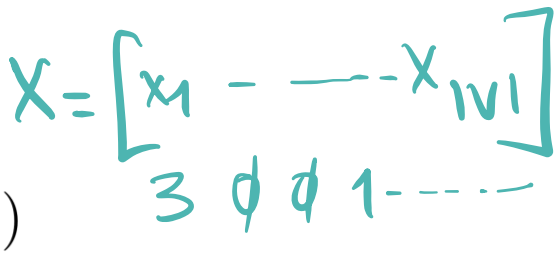
$$\texttt{Likelihoods:}\mathbf{P}\left(w_t \,|\, Y = y_k\right) \approx \frac{\sum_{i=1}^{N} X_{i,t} z_{i,k}}{\sum_{t'=1}^{|V|} \sum_{i=1}^{N} X_{i,t'} z_{i,k}}$$

The relative frequency of $w_t$ in documents of class $Y = y_k$ with respect to the total number of words in documents of that class

**Classify new document $D$ with feature vector $X$:**

$$\mathbf{P}\left(Y = y_k \,|\, X\right) \propto \mathbf{P}\left(Y = y_k\right) \mathbf{P}\left(X \,|\, Y = y_k\right)$$

$$\mathbf{P}\left(X \,|\, Y = y_k\right) = \prod_{t=1}^{|V|} \mathbf{P}\left(w_t \,|\, Y = y_k\right)^{X_t}$$



# Classification with Multinomial Model

- Estimate $P(w_t|Y=y_k)$ as:

The relative frequency of $w_t$ in documents of class $Y = y_k$ with respect to the total number of words in documents of that class

**Classify new document $D$ with feature vector $X$:**

$$\mathbf{P}(Y = y_k \mid X) \propto \mathbf{P}(Y = y_k)\,\mathbf{P}(X \mid Y = y_k)$$

$$\mathbf{P}(X \mid Y = y_k) = \prod_{t=1}^{|V|} \mathbf{P}(w_t \mid Y = y_k)^{X_t}$$

- Example (notice that words not appearing in $X^{new}$ does not affect classification):

$$\mathbf{X}^{new} = \begin{bmatrix} 3 & 1 & 0 & 0 \end{bmatrix}$$

$$P(\begin{bmatrix} 3 & 1 & 0 & 0 \end{bmatrix} \mid S) = P(w_1 \mid S)^3 \times P(w_2 \mid S)^1 \times P(w_3 \mid S)^0 \times P(w_4 \mid S)^0$$

# Smoothing

- Simplified notation:
  - $T_{ct}$: the number of times a term t occurs in class c
  - P(t | c) : Probability of seeing the term t in class c

$$P(t \mid c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

  - Classify a document **X**, where each term t appears $X_t$ times, considering:

$$P(c \mid X) \propto P(c)\, P(X \mid c)$$

$$P(X \mid c) = \prod_{t=1}^{|V|} P(t \mid c)^{X_t}$$

# Example

- In this toy example, we will build a NB classifier with the multinomial document model, to learn to classify whether a given document is about “China” (c) or “not China”. We will use Laplace smoothing.
- We have the following training data:

| | docID | | c = China? |
|---|---|---|---|
| Training set | 1 | Chinese Beijing Chinese | Yes |
| | 2 | Chinese Chinese Shangai | Yes |
| | 3 | Chinese Macao | Yes |
| | 4 | Tokyo Japan Chinese | No |

Vocabulary = V = {Beijing, Chinese, Japan, Macao, Tokyo, Shangai}

N = 4 $\hat{P}(c) = 3/4$ $\hat{P}(\bar{c}) = 1/4$

c: $y_1$ = China (doc. is about China)

$\bar{c}$: $y_2$ = Not about China ( " not about " )

# Example - w/o smoothing

| | docID | | c = China? |
|---|---|---|---|
| Training set | 1 | Chinese Beijing Chinese | Yes |
| | 2 | Chinese Chinese Shangai | Yes |
| | 3 | Chinese Macao | Yes |
| | 4 | Tokyo Japan Chinese | No |
| Test sample d | 5 | Chinese Chinese Chinese Tokyo Japan | ? |

$V = \{Chinese, Beijing, Shangai, Macao, Tokyo, Japan\}$

Probability Estimation:

$$\hat{P}(\text{Chinese} \mid c) = 5/8$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = 0/8 = 0$$

$$\hat{P}(\text{Chinese} \mid \bar{c}) = 1/3$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = 1/3$$

Classification:

$$P(c \mid d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k \mid c)$$

$$P(c \mid d_5) \propto 3/4 \cdot (5/8)^3 \cdot 0^1 \cdot 0^1 = 0$$

$$P(\bar{c} \mid d_5) \propto 1/4 \cdot (1/3)^3 \cdot (1/3)^1 \cdot (1/3)^1 \approx 0.0000034$$

Follow / Study the example.

# Example w/ smoothing

| | docID | | c = China? |
|---|---|---|---|
| Training set | 1 | Chinese Beijing Chinese | Yes |
| | 2 | Chinese Chinese Shangai | Yes |
| | 3 | Chinese Macao | Yes |
| | 4 | Tokyo Japan Chinese | **No** |
| Test sample d | 5 | Chinese Chinese Chinese Tokyo Japan | ? |

Probability Estimation:

$$\hat{P}(\text{Chinese} \mid c) = (5+1)/(8+6) = 3/7$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (0+1)/(8+6) = 1/14$$

$$\hat{P}(\text{Chinese} \mid \bar{c}) = (1+1)/(3+6) = 2/9$$

$$\hat{P}(\text{Tokyo} \mid c) = \hat{P}(\text{Japan} \mid c) = (1+1)/(3+6) = 2/9$$

Classification:

$$P(c \mid d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k \mid c)$$

$$P(c \mid d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$P(\bar{c} \mid d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

# Smoothing

For each term, t, we need to estimate P(t|c)

$$\hat{P}(t \mid c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

$T_{ct}$ is the count of term t in all documents of class c

Because an estimate will be 0 if a term does not appear with a class in the training data, we need **smoothing**:

**Laplace Smoothing**

$$\hat{P}(t \mid c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + |V|}$$

|V| is the number of terms in the vocabulary



42

From **Information Retrieval** book, Prabakhar et al.

Available online: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

# Code

```
TRAINBERNOULLINB(ℂ, 𝔻)
1  V ← EXTRACTVOCABULARY(𝔻)
2  N ← COUNTDOCS(𝔻)
3  for each c ∈ ℂ
4  do N_c ← COUNTDOCSINCLASS(𝔻, c)
5     prior[c] ← N_c/N
6     for each t ∈ V
7     do N_ct ← COUNTDOCSINCLASSCONTAININGTERM(𝔻, c, t)
8        condprob[t][c] ← (N_ct + 1)/(N_c + 2)
9  return V, prior, condprob

APPLYBERNOULLINB(ℂ, V, prior, condprob, d)
1  V_d ← EXTRACTTERMSFROMDOC(V, d)
2  for each c ∈ ℂ
3  do score[c] ← log prior[c]
4     for each t ∈ V
5     do if t ∈ V_d
6           then score[c] += log condprob[t][c]
7           else score[c] += log(1 − condprob[t][c])
8  return arg max_{c∈ℂ} score[c]
```

► **Figure 13.3** NB algorithm (Bernoulli model): Training and testing. The add-one smoothing in Line 8 (top) is in analogy to Equation (13.7) with $B = 2$.

TRAINMULTINOMIALNB($\mathbb{C}, \mathbb{D}$)

1 $V \leftarrow$ EXTRACTVOCABULARY($\mathbb{D}$)
2 $N \leftarrow$ COUNTDOCS($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ COUNTDOCSINCLASS($\mathbb{D}, c$)
5 $prior[c] \leftarrow N_c / N$
6 $text_c \leftarrow$ CONCATENATETEXTOFALLDOCSINCLASS($\mathbb{D}, c$)
7 **for each** $t \in V$
8 **do** $T_{ct} \leftarrow$ COUNTTOKENSOFTERM($text_c, t$)
9 **for each** $t \in V$
10 **do** $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11 **return** $V, prior, condprob$

APPLYMULTINOMIALNB($\mathbb{C}, V, prior, condprob, d$)

1 $W \leftarrow$ EXTRACTTOKENSFROMDOC($V, d$)
2 **for each** $c \in \mathbb{C}$
3 **do** $score[c] \leftarrow \log prior[c]$
4 **for each** $t \in W$
5 **do** $score[c] \mathrel{+}= \log condprob[t][c]$
6 **return** $\arg\max_{c \in \mathbb{C}} score[c]$

► **Figure 13.2** Naive Bayes algorithm (multinomial model): Training and testing.

# Tf-idf

- The multinomial model uses a feature vector based solely on how **often** words appear in a document.

- **Very common words dominate.**
  Words like *"the"*, *"and"*, *"have"*, *"is"* appear frequently in almost every document, so raw counts don't help distinguish between documents or topics.

- **Rare but meaningful words get undervalued.**
  A word that appears only a few times — like *"offside"*,— may actually be a strong indicator of the document's subject.

- Now we will see TF-IDF features that are commonly used in text processing, to address these issues.

# Need for transforming tf counts

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- Imagine finding relevant documents in response to a query:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase **linearly** with term frequency.

# Term Document Frequencies

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{v}$: a column below

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 157 | 73 | 0 | 0 | 0 | 0 |
| Brutus | 4 | 157 | 0 | 1 | 0 | 0 |
| Caesar | 232 | 227 | 0 | 2 | 1 | 1 |
| Calpurnia | 0 | 10 | 0 | 0 | 0 | 0 |
| Cleopatra | 57 | 0 | 0 | 0 | 0 | 0 |
| mercy | 2 | 0 | 3 | 5 | 5 | 1 |
| worser | 2 | 0 | 1 | 1 | 1 | 0 |

3

# Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

# Inverse Document Frequency

- Rare terms are more informative than frequent terms
  - Recall stop words

- We will use the inverse document frequency (idf) to capture this.

# Inverse Document Frequency

- **$df_t$ is the document frequency of *t*: the number of documents that contain *t***
  - $df_t$ is an inverse measure of the informativeness of *t*
  - $df_t$ <= *N* *(the total number of documents in the training corpus)*

- We define the idf(inverse document frequency) of *t* by

$$idf_t = \log_{10}(N/df_t)$$

  - We use **log** ($N/df_t$) instead of $N/df_t$ to "dampen" the effect of idf.
  - The base of the log is not important.

- Note: There is one idf value for each term *t* in a collection.

# Tf-Idf

- **The tf-idf weight of a term in a document is the product of its tf weight and its idf weight.**

$$w_{t,d} = \log(1 + \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

  - Increases with the number of occurrences within a document
  - Increases with the rarity of the term in the collection

- **Summary**: TF-IDF produces one fixed-length vector per document, where the vector length equals the vocabulary size $|V|$.
  - Most dimensions are zero, making the representation sparse.

You are not responsible from the Text Processing slides (they are given as extra info). Just read for basic understanding of issues in NLP.

# Text Processing for Classification

# Preprocessing

- **Token normalization**
  - Remove superficial character variances from words
  - A.B.D , ABD => ABD
- **Stop-word removal**
  - Remove predefined common words that are not specific or discriminatory to the different classes
  - Remove a, the, …
- **Stemming**
  - Reduce different forms of the same word into a single word (base/root form)
  - Gidiyorum, gidiyorsun, gidiyor => git
- **Feature selection**
  - Choose feature that looks more relevant

# Tokenization

- Breaking the sentence into individual **tokens:**
  - "I LOVED the movie!!! It was AMAZING, but a little long."
  - ["I", "LOVED", "the", "movie", "!!!", "It", "was", "AMAZING", ",", "but", "a", "little", "long", "."]
- Traditional tokenization splits at spaces and punctuation.

# Tokenization

- Issues of tokenization are language specific
  - Requires the language to be known

    German compound nouns

  - East Asian Languages (Chinese, Japanese, Korean, Thai)
    - Text is written without any spaces between words

# Token Normalization

Remove superficial character variances from words

- **Lowercasing**
  - To avoid treating "Loved" and "loved" as different words.
- **Removing Punctuation**
  - Punctuation usually does not help (unless doing sentiment analysis with emojis, sarcasm, etc.)

car. => car

- A.B.D , ABD, => ABD
  - Anti-discriminatory => antidiscriminatory

# Token Normalization

Preprocessing is domain/problem specific

- How to handle special cases involving apostrophes, hyphens etc?
- C++, C#, URLs, emails, phone numbers, dates, compound words or word sequences like San Francisco, Los Angeles

# Text Preprocessing Using Python

**Text:** The leaves fall gently to the GROUND, as the wind scatters the leaves across the field.

- **Lowercase the sentence and remove punctuations**

```python
# Step 1: Lowercase and remove punctuation
import string
text = "The leaves fall gently to the GROUND, as the wind scatters the leaves across the field."
text = text.lower().translate(str.maketrans('', '', string.punctuation))

# Output: the leaves fall gently to the ground as the wind scatters the leaves across the field
```

- string.punctuation contains all standard punctuation characters and can be used to filter them from text.
- Alternatively, regular expressions can also be used to remove punctuation effectively.

**Output Text:** the leaves fall gently to the ground as the wind scatters the leaves across the field

# Stop Word Removal

- Very common words that have no discriminatory

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | an | and | are | as | at | be | by | for | from |
| has | he | in | is | it | its | of | on | that | the |
| to | was | were | will | with | | | | | |

► **Figure 2.5** A stop list of 25 semantically non-selective words which are common in Reuters-RCV1.

- Sort terms by collection frequency and take the most frequent words

- For an application, an **additional domain specific stopword** list may be constructed
  - In a collection about insurance practices, "insurance" would be a stop word

# Text Preprocessing Using Python

**Text:** the leaves fall gently to the ground as the wind scatters the leaves across the field

- **Tokenization and Stopword Removal**

```python
# Step 2: Tokenize and remove stopwords
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
tokens = word_tokenize(text)
stop_words = set(stopwords.words('english'))
tokens = [t for t in tokens if t not in stop_words]

# Output: ['leaves', 'fall', 'gently', 'ground', 'wind', 'scatters', 'leaves', 'across', 'field']
```

- The Natural Language Toolkit (NLTK) – a popular Python library provides **stopword lists for multiple languages**, which can be downloaded and used for filtering common words.
- It also offers various tokenizers; for example, the word tokenizer splits a sentence into individual words.

**Output Text:** ['leaves', 'fall', 'gently', 'ground', 'wind', 'scatters', 'leaves', 'across', 'field']

# Stemming & Lemmatization

Reducing words to their base form.

- **Stemming**: crude rule-based chopping

  "amazing" → "amaz"

- **Lemmatization**: uses grammar/dictionary (preferred but slower).
  - Produces **valid words.**
  - Tools such as Zemberek for Turkish

- Original words: "running", "runs", "better"
  - **Stemming** → run, run, better
  - **Lemmatization** → run, run, good

# Text Preprocessing Using Python

**Tokenized Text:** ['leaves', 'fall', 'gently', 'ground', 'wind', 'scatters', 'leaves', 'across', 'field']

- **Difference Between Stemming and Lemmatization**

```python
# Step 3: Stemming
from nltk.stem import PorterStemmer
nltk.download('wordnet')
stemmer = PorterStemmer()
stemmed = [stemmer.stem(t) for t in tokens]

# Output: ['leav', 'fall', 'gentli', 'ground', 'wind', 'scatter', 'leav', 'across', 'field']
```

```python
# Step 4: Lemmatization
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(t) for t in tokens]

# Output: ['leaf', 'fall', 'gently', 'ground', 'wind', 'scatter', 'leaf', 'across', 'field']
```

# Text Preprocessing Using Python

**Preprocessed Text:** ['leaf', 'fall', 'gently', 'ground', 'wind', 'scatter', 'leaf', 'across', 'field']

- **Calculating Counts & TFIDF Vectors**

```python
from sklearn.feature_extraction.text import CountVectorizer
doc = [" ".join(lemmatized)]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(doc)
print("Features:", vectorizer.get_feature_names_out())
print("Vector:", X.toarray())

# Features: ['across' 'fall' 'field' 'gently' 'ground' 'leaf' 'scatter' 'wind']
# Vector:   [[1       1      1       1        1       2       1        1]]
```

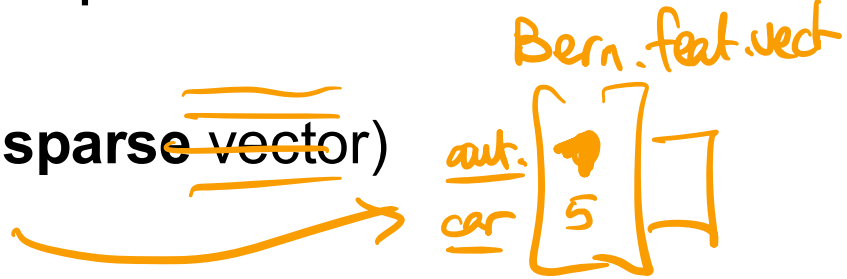
```python
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(doc)
print("Features:", tfidf.get_feature_names_out())
print("TF-IDF:", X_tfidf.toarray().round(3))

# Features: ['across' 'fall' 'field' 'gently' 'ground' 'leaf' 'scatter' 'wind']
# TF-IDF:   [[0.333  0.333  0.333  0.333   0.333   0.667   0.333   0.333]]
```

A completely new take
with Deep Learning

# Word Embeddings

# Motivation

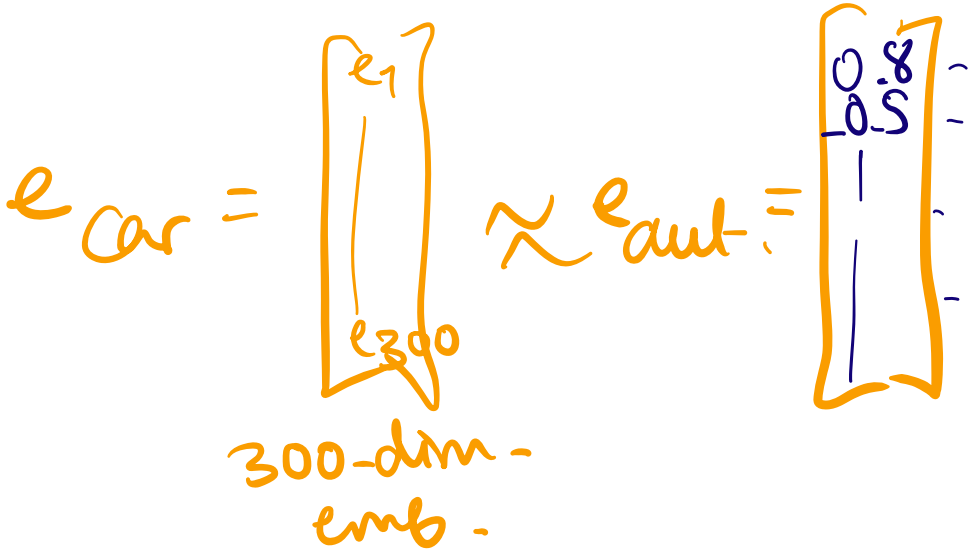
- So far we have seen a text was represented with vector of length vocabulary size |V|
  - Computational inefficiency (**huge**, **sparse** ~~vector~~)
  - **Semantic similarity** is ignored
    - Each word is as distant to another as to a similar word (e.g. car vs automobile) – each word is orthogonal to one another

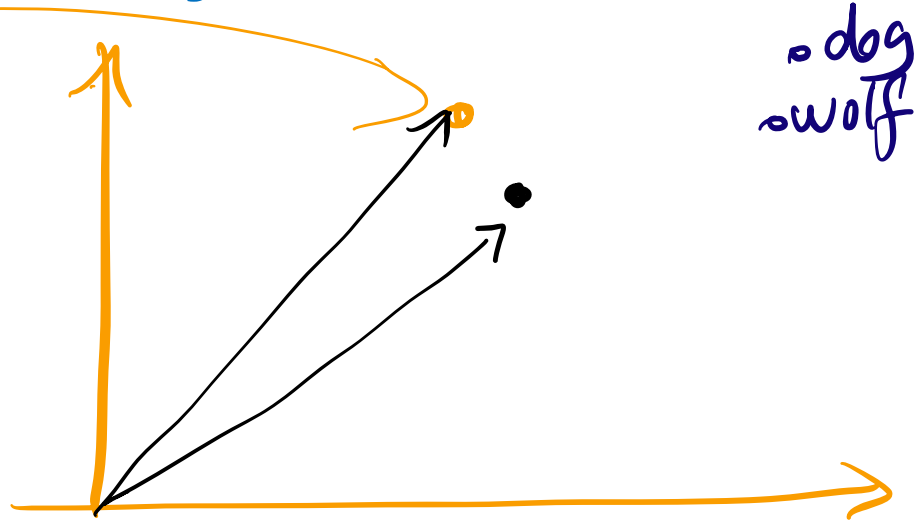



$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

one-hot repr.

- Can we have a richer representation which also encodes the **semantic similarity** between words?
  - car vs automobile
  - liked vs loved vs hated

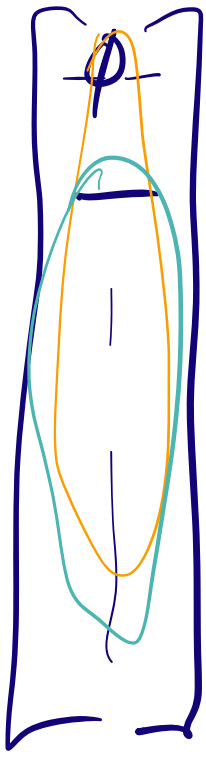



Compare
- dimensionality
  $|V| \approx 10{,}000^+ \rightarrow d = 300$
- sparse vectors → dense vector

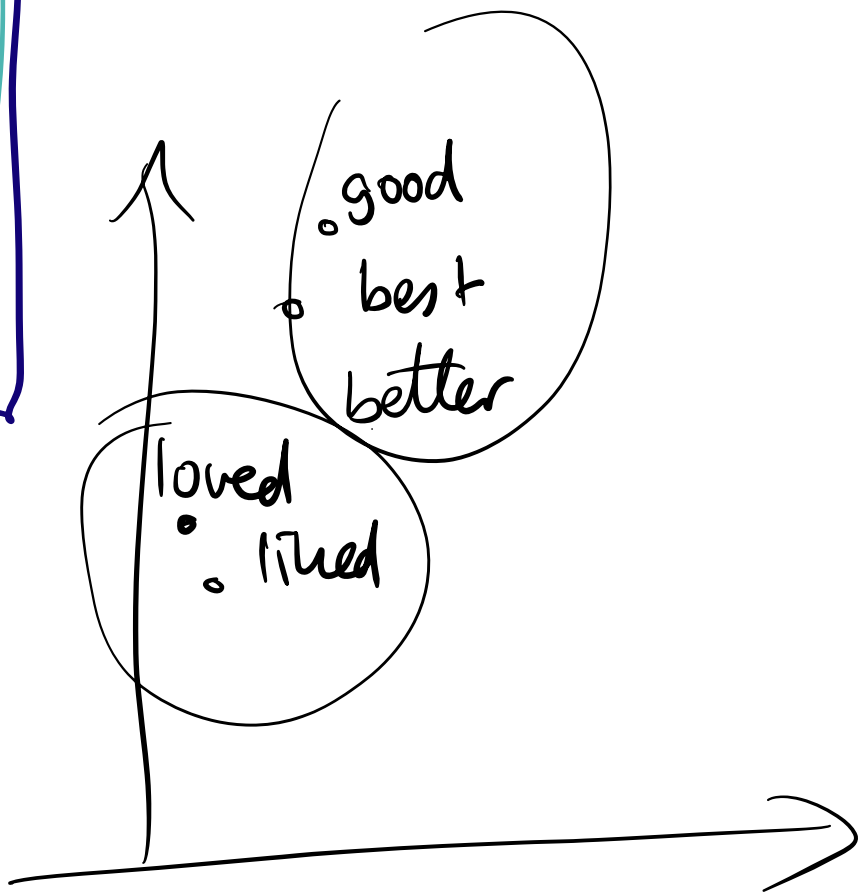
(word embeddings)

$e_{car} =$



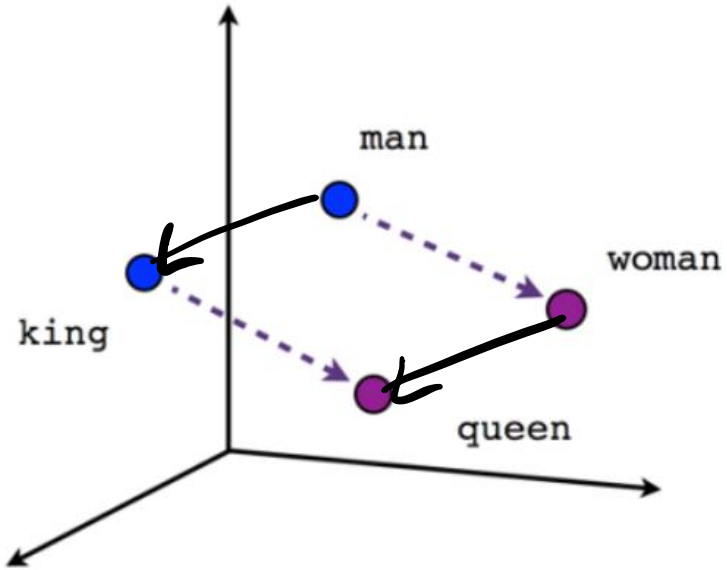
living? / no (0)
(1)

size - (human? / more / less)

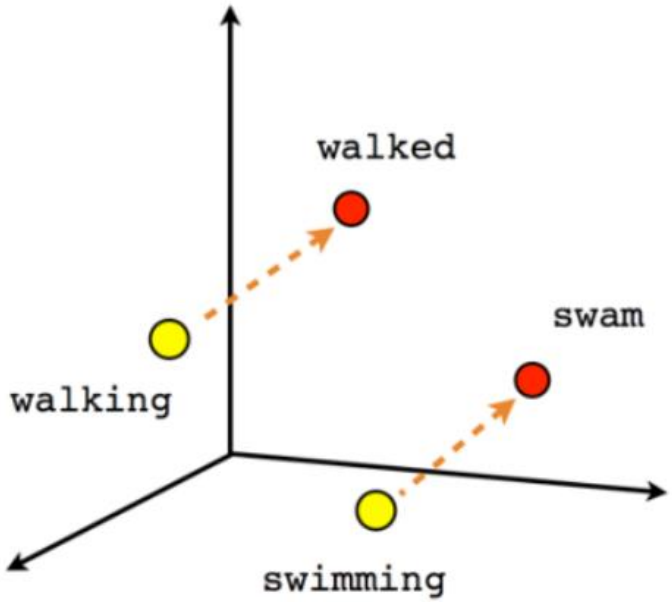


# Word Embeddings

- Word embeddings (like Word2Vec, GloVe, FastText) replace sparse count vectors with **dense vectors that encode meaning**.
  - Each word is mapped to a **fixed-length** vector (e.g., 300-dimensional)
    - You can think of the dimensions as attributes of a concept for intuition, but in reality semantic attributes are distributed (attributes and dimensions are not one-to-one)
  - Similar words are close in **vector space** (cosine similarity)
  - Learnt in **unsupervised** fashion, from a huge corpus




Male-Female

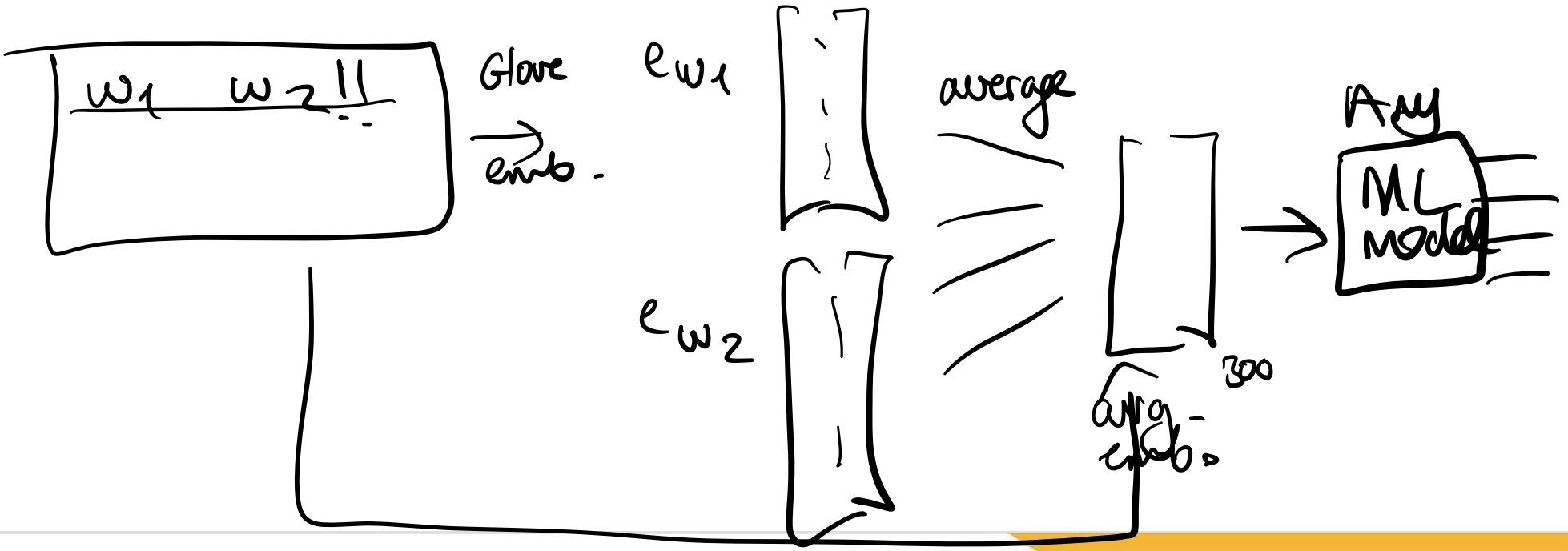



Verb tense

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

# Use of Word Embeddings

| Step | Representation |
| --- | --- |
| "This movie was absolutely amazing!" | →["this","movie","was","absolutely","amazing"] |
| Lookup GloVe vectors | → 5 vectors (each 100-dim) |
| Aggregation (e.g. averaging) | → 1 vector (100-dim) representing whole sentence |
| Classification | Logistic regression, NN, SVM… with 100-dimensional features |



- We will see how to learn word embeddings in the unsupervised learning section, but luckily there are some **pretrained embeddings**, obtained over very large corpuses.
  - E.g. https://nlp.stanford.edu/projects/glove/.