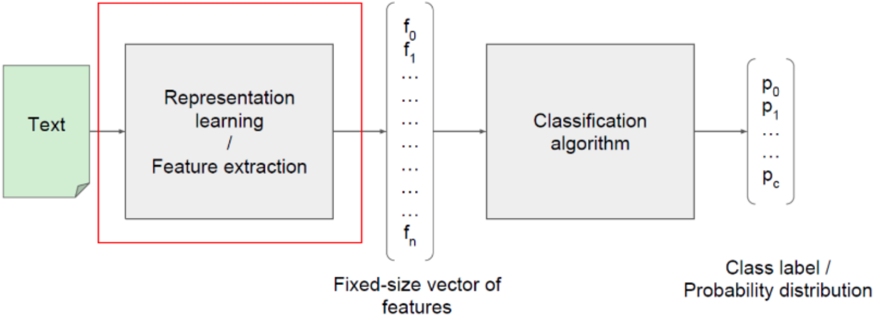
NATURAL LANGUAGE PROCESSING (PRACTICE)

NLP 242 - Lab 4: VECTOR SEMANTIC AND EMBEDDINGS

Department of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Vector Semantics and Embeddings

The system of text classification


Text
Representation learning / Feature extraction
f_0
f_1
...
f_n
Fixed-size vector of features
Classification algorithm
p_0
p_1
...
p_c
Class label / Probability distribution

Features

- A variable that can be measured, representing differences in something we want to model.
- We typically select features that are useful for identifying something, such as classification.
  - **Example:** The girl was **very beautiful** at the party.
- We usually need a sufficient number of features to build a comprehensive model.
  - *But not too many!*

Text Representation

Useful features can be manually engineered (feature engineering)

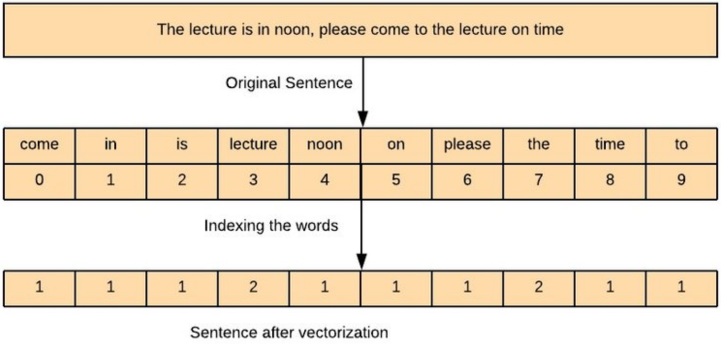
- Statistical features: length, position, etc.
- Using scores from dictionaries:
  - Sentiment dictionaries: SentiWordNet, SentiWords, etc.
  - Subjectivity/objectivity dictionaries: MPQA
- Syntactic features:
  - POS tags
- Task-specific features: e.g., number of emojis (:)) or :(()

Feature Vector

- **Value** of some features from an observation can be represented as a vector.
- **Features** will be useful in distinguishing between categories.

Count Vectorizing Features

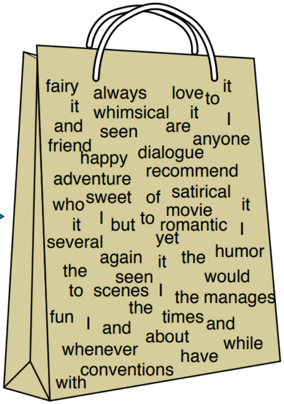
- Vectorization is the process of transforming text documents into numerical feature vectors.
- This specific strategy (tokenization, counting, and normalization) is called **Bag of Words**.
- CountVectorizing is used for this purpose.


The lecture is in noon, please come to the lecture on time
Original Sentence
Indexing the words
Sentence after vectorization

come	in	is	lecture	noon	on	please	the	time	to
0	1	2	3	4	5	6	7	8	9
1	1	1	2	1	1	1	2	1	1

Bag of Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

N-grams Features

- The Bag-of-Words (BOW) model does not effectively represent the order of words → use N-grams.
- The N-grams model generates features based on grams.

Example:

- "I am learning NLP"
- Uni-grams:
  - "I", "am", "learning", "NLP"
- Bi-grams:
  - "I am", "am learning", "learning NLP"

```
text = ["I love NLP and I will learn NLP in 2 months"]
{'love nlp': 3, 'nlp and': 4, 'and will': 0, 'will learn':
6,'learn nlp': 2, 'nlp in': 5, 'in 2 months': 1}
```

Co-occurrence Matrix

- Counts the co-occurrence of words in a matrix.
- **Co-occurrence word statistics** describe words that frequently appear together, representing relationships between words.
- The statistical process involves counting the occurrences of words in a text corpus.

Co-occurrence Matrix

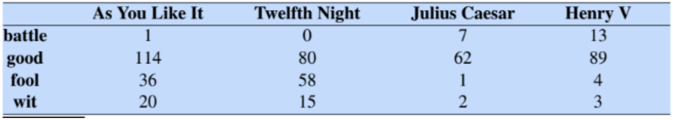
- I like deep learning.
- I like NLP.
- I enjoy flying.

	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	1	0	1	1	0
enjoy	1	1	0	0	0	0	1
deep	0	0	0	0	1	0	0
learning	0	1	0	1	0	0	0
NLP	0	1	0	0	0	0	1
flying	0	0	1	0	0	1	0

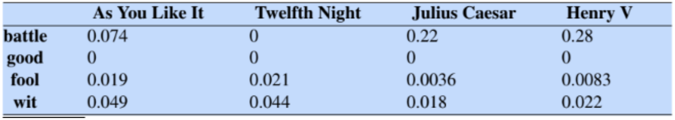
TF-IDF: Weighing terms in the vector

$$w_{i,j} = \mathbf{tf}_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

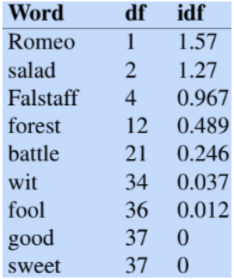
$$tf_{i,j} = \text{number of occurrences of } i \text{ in } j$$
$$df_i = \text{number of documents containing } i$$
$$N = \text{total number of documents}$$



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022



Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Naive Bayes

Bayes' Theorem

$$P(B|A) = \frac{P(AB)}{P(A)} = \frac{P(A|B)P(B)}{P(A)} = \frac{P(A|B)P(B)}{P(AB) + P(A\bar{B})}$$

- Posterior: $P(B \mid A)$
- Likelihood (Conditional): $P(A \mid B)$
- Prior: $P(B)$
- Evidence: $P(A)$

Posterior = Likelihood * Prior / Evidence

Naive Bayes Method

Naïve Bayes Assumption: The attributes that describe the data are conditionally independent given the classification hypothesis.

- Data point $x = (x_1, ..., x_n)$, label $y \in \{0, 1\}$
- Formulate a probabilistic model that places a distribution $P(x, y)$
- Compute $P(y|x)$, predict $\text{argmax}_y P(y|x)$ to classify

$$\begin{aligned} P(y|x) &= \frac{P(y)P(x|y)}{P(x)} (\textit{Bayes' Rule}) \\ &\propto P(y)P(x|y)(P(x) \textit{is constant: irrelevant for finding the max}) \\ &= P(y)\prod_{i=1}^{n} P(x_i|y) \end{aligned}$$

$$\text{argmax}_y P(y|x) = \text{argmax}_y \log P(y \mid x) = \text{argmax}_y \left[\log P(y) + \sum_{i=1}^{n} \log P(x_i|y)\right]$$

Maximum Likelihood Estimation

- **Data points** (x_j, y_j) provided (j indexes over examples).
- Find values of $P(y)$, $P(x|y)$ that maximize data likelihood (generative):

$$\prod_{j=1}^{m} P(y_j, x_j) = \prod_{j=1}^{m} P(y_j) \left[\prod_{i=1}^{n} P(x_{ji}|y_j) \right]$$

- $j \rightarrow$ Data points, $i \rightarrow$ Features
- $x_{ji} \rightarrow$ i-th feature of the j-th example

Equivalent to maximizing the logarithm of data likelihood:

$$\sum_{j=1}^{m} \log P(y_j, x_j) = \sum_{j=1}^{m} \log P(y_j) + \sum_{j=1}^{m} \sum_{i=1}^{n} \log P(x_{ji}|y_j)$$

Naïve Bayes Algorithm

Objective: Estimate the posterior probability $P(\text{document}|\text{class})$ and the prior probability $P(\text{class})$.

Likelihood: Assume the *bag of words* model

- A document is a sequence of words $(w_1, \ldots, w_n)$.
- The order of words is not important.
- Each word is conditionally independent given the class of the document:

$$P(\text{document}|\text{class}) = P(w_1, \ldots, w_n|\text{class}) = \prod_{i=1}^{n} P(w_i|\text{class})$$

- This simplifies the problem to estimating the probability of each word $P(w_i|\text{class})$.

Parameter Estimation

- Model parameters: likelihood probabilities $P(\text{word}|\text{class})$ and prior probabilities $P(\text{class})$.
- How do we determine the values of these parameters?

prior
spam: 0.33
¬spam: 0.67

P(word | spam)	
the	0.0156
to	0.0153
and	0.0115
you	0.0093
a	0.0086
with	0.0080
from	0.0075

P(word | ¬spam)	
the	0.0210
to	0.0133
of	0.0119
with	0.0108
from	0.0107
and	0.0105
a	0.0100

Parameter Estimation

Model parameters: likelihood probabilities $P(\text{word}|\text{class})$ and prior probabilities $P(\text{class})$.

- How do we determine the values of these parameters?
- Training data is required.

$$P(\text{word}|\text{class}) = \frac{\text{\# of occurrences of this word in docs from this class}}{\text{total \# of words in docs from this class}}$$

- The maximum likelihood (ML) estimation process from training data is:

$$\prod_{d=1}^{D} \prod_{i=1}^{n_d} P(w_{d,i}|\text{class}_{d,i})$$

where d: index of training document, i: index of a word.

Laplacian Smoothing

- **Parameter smoothing**: Handling cases where words do not appear or appear too frequently.
- **Laplacian smoothing**:

$$P(\text{word}|\text{class}) = \frac{\#\text{ of occurrences of this word in docs from this class} + 1}{\text{total }\#\text{ of words in docs from this class} + V}$$

where V total number of unique words.

Issues with Naive Bayes

*The film was **beautiful**, **stunning** cinematography and **gorgeous** sets, but **boring***

$$\begin{aligned} P(x_{\text{beautiful}}|+) &= 0.1 & P(x_{\text{beautiful}}|-) &= 0.01 \\ P(x_{\text{stunning}}|+) &= 0.1 & P(x_{\text{stunning}}|-) &= 0.01 \\ P(x_{\text{gorgeous}}|+) &= 0.1 & P(x_{\text{gorgeous}}|-) &= 0.01 \\ P(x_{\text{boring}}|+) &= 0.01 & P(x_{\text{boring}}|-) &= 0.1 \end{aligned}$$

Independence assumption issue

- *Beautiful* and *gorgeous* are dependent.

Simple assumption

- Computing $P(x, y)$ via $P(y|x)$.

Discriminative models directly compute $P(y|x)$

Spam filtering

What is Spam?

- Spam is the use of electronic messaging systems to send unwanted messages (spam mail),
- Especially advertisements,
- As well as sending repeated messages on the same webpage.

What is Spam Filtering?

- A type of text classification problem,
- Classifying emails as spam or normal (ham),
- Example: The spam folder in a **Gmail** account,
- Common email datasets used: **Ling Spam Corpus** & **Enron Spam Corpus**.

Spam Email Data

Enron Email Dataset

- The dataset contains 33,716 emails across 6 folders.
- Each folder contains emails classified as either normal (‘ham’) or spam (‘spam’).
- The total number of spam and normal emails are 16,545 and 17,171, respectively.

Example

Email Ham (Normal)
Subject: re : indian springs this deal is to book the teco pvr revenue. It is my understanding that teco just sends us a check, I haven't received an answer as to whether there is a predetermined price associated with this deal or if teco just lets us know what we are giving. I can continue to chase this deal down if you need.

Email Spam
Subject: photoshop, windows, office, cheap. main trending abasements darer prudently fortuitous undergone lighthearted charm orinoco taster railroad affluent pornographic cuvier irvin parkhouse blameworthy chlorophyll robed diagrammatic fogarty clears bayda inconveniencing managing represented smartness hashish academics shareholders unload badness danielson pure caffein spaniard chargeable levin.

Evaluation

Evaluating Classifiers: How well does our classifier work? Let's first address binary classifiers:

- Is this email spam?
  spam (+) or not spam (-)
- Is this post about Delicious Pie Company?
  about Del. Pie Co (+) or not about Del. Pie Co (-)

First step in evaluation: The confusion matrix

	gold standard labels	
	gold positive	gold negative
system positive	true positive	false positive
system negative	false negative	true negative

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

Why don't we use accuracy?

Ex 1: Accuracy of our **"nothing is pie"** classifier

- 999,900 true negatives and 100 false negatives
- Accuracy is $\frac{999{,}900}{1{,}000{,}000} = \mathbf{99.99\%}$!
- But useless at finding pie-lovers (or haters)!! **Which was our goal!**

$\Rightarrow$ Accuracy doesn't work well for unbalanced classes **Most tweets are not about pie!**

Ex 2: Optimal cost to solve the problem. Costs for Patients

- Positive for Coronavirus: 20 dollars.
- Negative for Coronavirus: 5 dollars.
- On average, one patient will infect 5 other patients.

Precision and Recall and F1- score

	gold positive	gold negative
system positive	true positive	false positive
system negative	false negative	true negative

- Precision: % of selected items that are correct: $TP/(TP+FP)$.
- Recall: % of correct items that are selected: $TP/(TP+FN)$.

Just say no: every tweet is **"not about pie"**: 100 tweets talk about pie, 999,900 tweets don't.

$$Accuracy = 999,900/1,000,000 = \mathbf{99.99\%}$$

But the **Recall** and **Precision** for this classifier are terrible:

$$Recall = \frac{TP}{TP+FN} =? \text{ and } Precision = \frac{TP}{TP+FP} =?$$

F_1 score is the (weighted) harmonic mean of precision and recall: $F_1 = \frac{2PR}{P+R}$

F1 Score and Multi classes

We can define precision and recall for multiple classes like this

	Urgent	**Normal**	**Spam**
Urgent	8	10	1
Normal	5	60	50
Spam	3	30	200

$$\text{recall}_u = \frac{8}{8+5+3} \quad \text{recall}_n = \frac{60}{10+60+30} \quad \text{recall}_s = \frac{200}{1+50+200}$$

$$\text{precision}_u = \frac{8}{8+10+1} \quad \text{precision}_n = \frac{60}{5+60+50} \quad \text{precision}_s = \frac{200}{3+30+200}$$

$$\text{microaverage precision} = \frac{8+60+200}{268+99}$$

$$\text{macroaverage precision} = \frac{0.42+0.52+0.86}{3}$$

THANKS FOR LISTENING!