NLP501 - MASTER OF SOFTWARE ENGINEERING

Natural Language Processing

Session 01: Introduction & Sentiment Analysis

Logistic Regression for Text Classification

Today's Agenda

PART 1

Introduction to NLP

Definition, history, applications, and challenges

PART 2

Text Preprocessing

Tokenization, normalization, stemming, lemmatization

PART 3

Feature Extraction

Bag of Words, TF-IDF, N-grams

PART 4

Sentiment Analysis with Logistic Regression

Mathematical foundations and implementation

PART 1: INTRODUCTION

What is Natural Language Processing?

Natural Language Processing (NLP) is a field at the intersection of **computer science**, **artificial intelligence**, and **linguistics** that enables computers to understand, interpret, and generate human language.

Understanding

Comprehend meaning, context, and intent from text or speech

Interpretation

Extract structured information and relationships from unstructured text

Generation

Produce human-like text, responses, and translations

Evolution of NLP

1950s-1960s

Rule-Based Era

Georgetown experiment, Chomsky's grammar theories

→

1970s-1980s

Statistical Methods

Hidden Markov Models, probabilistic parsing

→

1990s-2000s

Machine Learning

SVM, Naive Bayes, Maximum Entropy

→

2010s-Present

Deep Learning

Word2Vec, LSTM, Transformers, GPT, BERT

Key Milestones

1950

Turing Test proposed

2013

Word2Vec introduced by Google

2017

Transformer architecture "Attention Is All You Need"

Real-World NLP Applications

Search Engines
Query understanding, document ranking, semantic search

Sentiment Analysis
Brand monitoring, customer feedback, market research

Email Filtering
Spam detection, priority inbox, smart categorization

Virtual Assistants
Siri, Alexa, Google Assistant - intent recognition

Text Summarization
News aggregation, document summarization, meeting notes

Healthcare
Clinical note analysis, drug discovery, medical coding

Machine Translation
Google Translate, DeepL - cross-language communication

Chatbots
Customer service, FAQ automation, conversational AI

Finance
Trading signals, risk assessment, fraud detection

Why is NLP Difficult?

Ambiguity

"I saw the man with the telescope"

Who has the telescope? The speaker or the man?

Language Variability

Dialects, slang, misspellings, abbreviations

"gonna", "wanna", "u r gr8"

Context Dependence

"Apple released a new product"

Fruit company or tech company?

Domain Specificity

Medical, legal, technical jargon

"The patient presents with acute MI"

Sarcasm and Irony

"Oh great, another meeting!"

Positive words, negative sentiment

Implicit Knowledge

Common sense reasoning required

"The trophy doesn't fit in the suitcase because it's too big"

PART 1: INTRODUCTION

The NLP Pipeline

Raw Text Input data → **Preprocessing** Clean & normalize → **Feature Extraction** Vectorization → **Model** ML/DL algorithm → **Output** Prediction

Example: Sentiment Classification Pipeline

INPUT
"I loved this movie! It was amazing."
→
TOKENS
["loved", "movie", "amazing"]
→
VECTOR
[0, 1, 0, 1, ..., 1]
→
OUTPUT
Positive ✓

PART 2

Text Preprocessing

Cleaning and preparing text data for analysis

Preprocessing Steps

1

Tokenization

Split text into words/tokens

2

Case Folding

Convert to lowercase

3

Noise Removal

Remove punctuation, special chars

4

Stop Words

Remove common words

5

Stemming

Reduce to word stems

6

Lemmatization

Reduce to base form

Why Preprocess?

- Reduce vocabulary size
- Remove noise and irrelevant information
- Normalize variations of same word
- Improve model performance
- Reduce computational cost

Tokenization

Definition

Breaking text into smaller units called **tokens** (words, subwords, or characters)

Example

Input:

```
"Hello, world! How are you?"
```

Output (Word Tokenization):

```
["Hello", ",", "world", "!", "How", "are", "you", "?"]
```

NLTK Code

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
```

Word Tokenization

Split by whitespace and punctuation

Sentence Tokenization

Split text into sentences

Subword Tokenization

BPE, WordPiece (used in BERT)

Character Tokenization

Each character is a token

Stop Words Removal

What are Stop Words?

Common words that carry little semantic meaning: **the, is, at, which, on, a, an, and, or, but, in, of, to**

Example

Before:

```
"The movie was not good at all"
```

After:

```
["movie", "good"]
```

NLTK Code

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
filtered = [w for w in tokens if w not in stop_words]
```

⚠ Caution

"not" is often a stop word but crucial for sentiment!

"not good" → "good" loses negation

When to Remove Stop Words?

✓ Document classification

✓ Information retrieval

✓ Topic modeling

✗ Sentiment analysis (be careful)

✗ Machine translation

✗ Question answering

Stemming vs Lemmatization

Stemming

Chops off word endings using heuristic rules. Fast but crude.

Examples (Porter Stemmer):

```
running → runn
studies → studi
better → better
university → univers ✗
```

✓ Fast processing
✓ No dictionary needed
✗ May produce non-words

Lemmatization

Reduces words to dictionary base form (lemma). Uses morphological analysis.

Examples (WordNet):

```
running → run
studies → study
better → good
university → university ✓
```

✓ Produces valid words
✓ More accurate
✗ Slower, needs POS tags

Complete Preprocessing Pipeline

```
import nltk, re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(t) for t in tokens]
    return tokens
```

Example:

"The movies were AMAZING!!!" ---> ['movie', 'amazing']

PART 3

Feature Extraction

Converting text to numerical representations

Bag of Words (BoW)

Concept

Represent text as a vector of word counts. Ignores word order and grammar - treats document as a "bag" of words.

Example

Doc 1: "I love NLP"

Doc 2: "I love machine learning"

Vocabulary: [I, love, NLP, machine, learning]

```
Doc 1: [1, 1, 1, 0, 0]
Doc 2: [1, 1, 0, 1, 1]
```

Advantages

- ✓ Simple to implement
- ✓ Works well for many tasks
- ✓ Efficient computation

Limitations

- ✗ Loses word order
- ✗ High dimensionality
- ✗ Sparse vectors
- ✗ No semantic meaning

TF-IDF: Term Frequency-Inverse Document Frequency

Term Frequency (TF)

How often a term appears in a document

$$TF(t,d) = count(t \text{ in } d) \,/\, \text{total terms in } d$$

Inverse Document Frequency (IDF)

How rare/important a term is across documents

$$IDF(t) = \log(N \,/\, df(t))$$

N = total documents, df(t) = docs containing t

TF-IDF Score

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

High TF-IDF = important term for this document

Key Insight

- Words appearing in **many** documents get **lower** weight
- Words appearing in **few** documents get **higher** weight
- Common words like "the" have low TF-IDF

TF-IDF: Worked Example

Corpus (N = 3 documents)

Doc 1: "the cat sat on the mat"

Doc 2: "the dog sat on the log"

Doc 3: "the cat chased the dog"

Calculate TF-IDF for "cat" in Doc 1

Step 1: TF("cat", Doc1)

= 1/6 = 0.167

Step 2: IDF("cat")

= log(3/2) = 0.176

Step 3: TF-IDF

= 0.167 × 0.176 = 0.029

Calculate TF-IDF for "the" in Doc 1

Step 1: TF("the", Doc1)

= 2/6 = 0.333

Step 2: IDF("the")

= log(3/3) = 0

Step 3: TF-IDF

= 0.333 × 0 = 0

Key Observation

"the" appears in ALL documents → IDF = 0

"cat" appears in 2/3 docs → IDF > 0

Discriminative words get higher weights!

N-grams: Capturing Word Context

Definition

N-grams are contiguous sequences of N items (words or characters) from text. They help capture local word order and context.

Example: "I love machine learning"

Unigrams (n=1):

```
["I", "love", "machine", "learning"]
```

Bigrams (n=2):

```
["I love", "love machine", "machine learning"]
```

Trigrams (n=3):

```
["I love machine", "love machine learning"]
```

Why Use N-grams?

- Capture phrases: "not good" vs "good"
- Detect negations and modifiers
- Improve classification accuracy

Trade-offs

- Higher N = more context but sparser data
- Common: unigrams + bigrams combined

Sentiment Example

"not good" as bigram → negative

"not" + "good" as unigrams → confusing

PART 4

Sentiment Analysis with Logistic Regression

Classifying text as positive or negative

What is Sentiment Analysis?

Sentiment Analysis (or Opinion Mining) is the task of **automatically detecting** the emotional tone, attitudes, or opinions expressed in text.

😊
Positive
"Great product!"

😐
Neutral
"It's okay"

😠
Negative
"Terrible service"

Applications

- 📊 Brand monitoring and reputation management
- 🛒 Product review analysis
- 📈 Stock market prediction
- 🗳️ Political opinion mining
- 🎬 Movie/restaurant reviews
- 🎧 Customer feedback analysis

Binary Classification Problem

Problem Formulation

Given a text document **x**, predict whether the sentiment is **positive (y=1)** or **negative (y=0)**

Training Data

Input:

Labeled examples $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

Example:

("This movie is amazing!", 1)

("Worst experience ever", 0)

Goal

Learn a function $f: X \rightarrow \{0, 1\}$

More specifically:

Learn $P(y=1|x)$ - probability of positive sentiment given text x

Prediction

If $P(y=1|x) > 0.5$:

→ **Predict Positive**

If $P(y=1|x) \leq 0.5$:

→ **Predict Negative**

Why Not Linear Regression?

Linear Regression Output

$$y = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

Output can be any real number: -∞ to +∞

⚠️ Problem

- We need probability: $0 \leq P(y=1) \leq 1$
- Linear regression can give values like -2.5 or 3.7
- Not interpretable as probability!

Solution: Logistic Regression

Apply a function that squashes output to [0, 1]

$$P(y=1|x) = \sigma(w \cdot x + b)$$

where σ is the sigmoid function:

$$\sigma(z) = 1 / (1 + e^{-z})$$

The Sigmoid Function

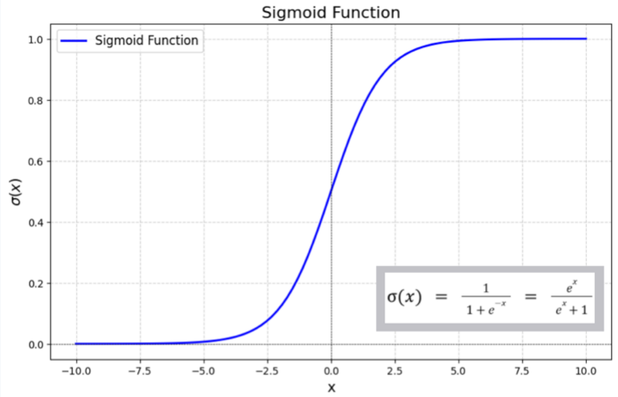
Definition

$$\sigma(z) = 1 / (1 + e^{-z})$$

Key Properties

- **Range:** Output always between 0 and 1
- **$\sigma(0) = 0.5$:** Decision boundary
- **$\sigma(z)$** → **1** as z → +∞
- **$\sigma(z)$** → **0** as z → -∞
- **Smooth and differentiable**

Sigmoid Curve (S-shaped)



z = 2
$\sigma(2) = 0.88$

z = 0
$\sigma(0) = 0.50$

z = -2
$\sigma(-2) = 0.12$

Logistic Regression Model

Model Equation

Step 1: Linear combination

$$z = w \cdot x + b = \Sigma_i\, w_i x_i + b$$

→

Step 2: Apply sigmoid

$$\hat{y} = \sigma(z) = 1/(1+e^{-z})$$

Parameters

w = weight vector $[w_1, w_2, \ldots, w_n]$

b = bias term (intercept)

x = feature vector (e.g., TF-IDF)

Interpretation

- **$w_i > 0$:** feature increases P(positive)
- **$w_i < 0$:** feature decreases P(positive)
- **$|w_i|$ large:** feature is important

Decision Rule

If $\hat{y} \geq 0.5$: predict **Positive**

If $\hat{y} < 0.5$: predict **Negative**

(threshold can be adjusted)

Loss Function: Cross-Entropy

Goal: Measure how wrong our predictions are

We need a function that penalizes confident wrong predictions more than uncertain ones.

Binary Cross-Entropy Loss (for single example)

$$L(y, \hat{y}) = -[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$$

where y = true label (0 or 1), $\hat{y}$ = predicted probability

When y = 1 (Positive)

$L = -\log(\hat{y})$

If $\hat{y} \approx 1 \rightarrow L \approx 0$ (good)
If $\hat{y} \approx 0 \rightarrow L \rightarrow \infty$ (bad)

When y = 0 (Negative)

$L = -\log(1-\hat{y})$

If $\hat{y} \approx 0 \rightarrow L \approx 0$ (good)
If $\hat{y} \approx 1 \rightarrow L \rightarrow \infty$ (bad)

Total Loss (m examples)

$J(w,b) = -1/m \ \Sigma_i \ L(y_i, \hat{y}_i)$

Average over all training examples

Training: Gradient Descent

Goal: Find w, b that minimize J(w, b)

Iteratively update parameters in the direction of steepest descent (negative gradient)

Update Rules

$$w_j := w_j - \alpha \cdot \partial J/\partial w_j$$

$$b := b - \alpha \cdot \partial J/\partial b$$

where α = learning rate (hyperparameter)

Gradients (Derivatives)

$$\partial J/\partial w_j = 1/m\, \Sigma_i\, (\hat{y}_i - y_i) \cdot x_{ij}$$

$$\partial J/\partial b = 1/m\, \Sigma_i\, (\hat{y}_i - y_i)$$

Learning Rate α

- Too small → slow convergence
- Too large → may overshoot
- Typical: 0.001 to 0.1

Gradient Descent Algorithm

```
# Initialize parameters
w = zeros(n_features)
b = 0

# Training loop
for epoch in range(num_epochs):
    # Forward pass
    z = X @ w + b
    y_pred = sigmoid(z)

    # Compute gradients
    dw = (1/m) * X.T @ (y_pred - y)
    db = (1/m) * sum(y_pred - y)

    # Update parameters
    w = w - alpha * dw
    b = b - alpha * db
```

Step 1: Initialize

Set weights to 0 or small random values

Step 2: Forward Pass

Compute predictions for all examples

Step 3: Compute Gradients

Calculate how loss changes with parameters

Step 4: Update

Move parameters in opposite direction of gradient

Regularization: Preventing Overfitting

Overfitting Problem

Model learns training data too well, including noise

- High accuracy on training data
- Poor generalization to new data

L1 Regularization (Lasso)

$$J\_reg = J + \lambda \cdot \Sigma_j |w_j|$$

Drives some weights to exactly 0 (feature selection)

L2 Regularization (Ridge)

$$J\_reg = J + \lambda \cdot \Sigma_j w_j^2$$

Penalizes large weights, keeps all features

Regularization Strength λ

- λ = 0: No regularization
- λ small: Light regularization
- λ large: Heavy regularization

Typical: 0.01, 0.1, 1.0 (tune via validation)

Model Evaluation Metrics

Why Not Just Accuracy?

Consider imbalanced datasets:

- Spam detection: 95% legitimate, 5% spam
- A model predicting "not spam" always gets 95% accuracy!
- But misses ALL actual spam emails

Key Metrics for Classification

Accuracy
Overall correctness

Precision
Quality of positives

Recall
Coverage of positives

F1 Score
Harmonic mean

Accuracy Formula

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Where:

- TP = True Positives (correctly predicted positive)
- TN = True Negatives (correctly predicted negative)
- FP = False Positives (Type I error)
- FN = False Negatives (Type II error)

PART 4: SENTIMENT ANALYSIS

Confusion Matrix

A table showing predicted vs actual classifications

ACTUAL \ PREDICTED	Positive	Negative
Positive	**TP** True Positive	**FN** False Negative
Negative	**FP** False Positive	**TN** True Negative

Sentiment Analysis Example

Task: Classify reviews as Positive/Negative

85 TP	15 FN
10 FP	90 TN

Accuracy = (85+90)/(85+90+10+15)

= 175/200 = 87.5%

Precision and Recall

Precision

"Of all predicted positives, how many are actually positive?"

$$\text{Precision} = TP \,/\, (TP + FP)$$

High Precision → Few false alarms

Example: 85/(85+10) = 89.5%

Recall (Sensitivity)

"Of all actual positives, how many did we catch?"

$$\text{Recall} = TP \,/\, (TP + FN)$$

High Recall → Few missed positives

Example: 85/(85+15) = 85%

When to prioritize Precision?

- Spam filter (avoid flagging legitimate emails)
- Search engines (show relevant results)
- Cost of false positive is high

When to prioritize Recall?

- Disease diagnosis (catch all sick patients)
- Fraud detection (catch all fraud cases)
- Cost of false negative is high

F1 Score: Balancing Precision & Recall

The Precision-Recall Trade-off

Improving one often hurts the other:

- More aggressive (lower threshold) → ↑ Recall, ↓ Precision
- More conservative (higher threshold) → ↑ Precision, ↓ Recall

F1 Score Formula

$$F1 = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Harmonic mean penalizes extreme values

Range: 0 (worst) to 1 (perfect)

Example Calculation

Precision = 89.5%

Recall = 85%

F1 = 2 × (0.895 × 0.85) / (0.895 + 0.85)

F1 = 0.872 = 87.2%

Why Harmonic Mean?

- Arithmetic mean of (90%, 10%) = 50%
- Harmonic mean of (90%, 10%) = 18%

→ **Forces both metrics to be high!**

PART 5

Case Study

Twitter Sentiment Analysis

1.6M Tweets

Binary Labels

Real Results

CASE STUDY: TWITTER SENTIMENT

Dataset: Sentiment140

Dataset Overview

1.6M	800K	800K
Total Tweets	Positive	Negative

Source: Stanford CS224n (Go et al., 2009)

Labeling Method: Distant Supervision

Used emoticons as noisy labels:

- :) :D → Positive
- :(→ Negative

Sample Tweets

POSITIVE

"Just had the best coffee ever! ☕ Starting my day right"

POSITIVE

"Love this new album! Been listening on repeat all day"

NEGATIVE

"Ugh another rainy Monday. Can't wait for this week to end"

NEGATIVE

"My flight got cancelled again. Worst airline ever!"

CASE STUDY: TWITTER SENTIMENT

Preprocessing Pipeline

Raw Tweet → Lowercase → Clean URLs → Tokenize → Stop/Stem → **Clean Tokens**

Step-by-Step Example

1. Raw Tweet:

"@user OMG!! Check this out https://t.co/abc 😍 SO AMAZING!!! #blessed"

2. Lowercase:

ÀĚ ÞŒŇŐŎÖ ŊǺŃOŇŃÔ POÒŒŎÞ P OPPŐŒĚCPĚŃŎ CMŃŃ 😍 ŒŎ MÖ MŖÒŊǺÁNŌŇŒŇÀ

3. Remove URLs, @mentions, punctuation:

"omg check this out so amazing blessed"

4. Tokenize + Remove Stopwords + Stem:

LÀŎ Ö ŊÀÆÀŃOŇŃÔÀÆÀVÖ MŖÀÆÀNŌŇŒÈĽ

Twitter-Specific Challenges

- @mentions and #hashtags
- Short URLs (t.co, bit.ly)
- Emojis and emoticons 😉 :)
- Slang: "lol", "brb", "omg"
- Character limit → abbreviations

Design Decisions

✓ Keep hashtag text (remove #)
✓ Remove emojis (used for labeling)
✓ Keep common slang words

CASE STUDY: TWITTER SENTIMENT

Feature Engineering

Feature Extraction Method

Count positive/negative word frequencies

$x = [1, \text{sum}(\text{freq}_p(w)), \text{sum}(\text{freq}_n(w))]$

Example: Feature Vector

Tweet: "I am happy because I love the movie"

$\Sigma\ \text{freq}_p = 3521 + 5200 = 8{,}721$

$\Sigma\ \text{freq}_n = 180 + 890 = 1{,}070$

x = [1, 8721, 1070]

Frequency Dictionary Example

happy: 3521 pos / 180 neg | sad: 140 pos / 2890 neg

love: 5200 pos / 890 neg | hate: 320 pos / 4100 neg

Why This Works?

Reduces V words to 3 features. Captures sentiment signal efficiently. Fast training with logistic regression.

CASE STUDY: TWITTER SENTIMENT

Results & Analysis

Model Performance

Accuracy	Precision	Recall
76.5%	77.2%	75.8%

F1 Score: 76.5%

Confusion Matrix (Test Set)

7,580 True Positive	2,420 False Negative
2,280 False Positive	7,720 True Negative

Learned Parameters

bias (b)	w_1 (pos_freq)	w_2 (neg_freq)
-0.05	+0.0003	-0.0003

Key Insights

- Simple features work surprisingly well!
- Errors often from sarcasm, irony
- More features → better accuracy

CASE STUDY: TWITTER SENTIMENT

Error Analysis: What Went Wrong?

Common Misclassifications

SARCASM / IRONY

"Oh great, another Monday. Just what I needed."

Predicted: Positive (word "great") | Actual: Negative

NEGATION

"This movie is not good at all."

Predicted: Positive (word "good") | Actual: Negative

MIXED SENTIMENT

"I love the design but hate the battery life."

Both positive and negative words present

CONTEXT-DEPENDENT

"This is sick!" (slang for amazing)

Predicted: Negative (word "sick") | Actual: Positive

Model Limitations

- Bag-of-words loses word order
- Cannot understand negation patterns
- No concept of sarcasm or tone
- Struggles with domain-specific slang

Potential Improvements

- N-grams (bigrams, trigrams)
- Negation handling
- Word embeddings
- Deep learning (RNN/LSTM)
- Transformers (BERT)

We'll explore these in future sessions!

PART 6

Lab Practice

Text Preprocessing & Feature Extraction

Preprocessing | Feature Extraction | Hands-on Coding

LAB PRACTICE

Today's Lab Tasks

1. Text Preprocessing Pipeline

Tokenization, stopword removal, stemming vs lemmatization, handle edge cases (URLs, mentions)

2. Feature Extraction

BoW with CountVectorizer, TF-IDF with TfidfVectorizer, analyze feature matrices

Required Libraries

nltk, sklearn.feature_extraction.text

Dataset: Sample tweets (1000 examples)

SUMMARY

Key Takeaways

NLP Fundamentals

- ✓ NLP bridges linguistics & computer science
- ✓ Language is inherently ambiguous & complex
- ✓ Statistical/ML approaches dominate modern NLP

Feature Extraction

BoW
Word counts

TF-IDF
Weighted importance

N-grams
Word sequences

TF-IDF handles common word bias better than BoW

Text Preprocessing

✓ Tokenization breaks text into units
✓ Stopword removal reduces noise
✓ Stemming/Lemmatization normalize words
✓ Preprocessing choices affect model performance

Logistic Regression for Sentiment

✓ Sigmoid maps to probability [0, 1]
✓ Cross-entropy loss for training
✓ Gradient descent optimizes parameters
✓ Regularization prevents overfitting