Sentiment Analysis using NLP

Example Text

Let's analyze the sentiment of this review:

"The movie was great, but the theater was uncomfortable."

Step 1: Preprocessing and Tokenization

First, we tokenize the sentence:

["The", "movie", "was", "great", ",", "but", "the", "theater", "was", "uncomfortable", "."]

We'll focus on content words and remove punctuation and stopwords:

["movie", "great", "theater", "uncomfortable"]

Step 2: Feature Extraction - Bag of Words (BoW)

Assume our vocabulary contains these words with assigned indices:

Word	**Index**
movie	0
great	1
theater	2
uncomfortable	3
excellent	4
terrible	5
enjoyed	6
boring	7

The BoW representation becomes:

[1, 1, 1, 1, 0, 0, 0, 0]

This means our sentence contains 1 occurrence each of "movie", "great", "theater", and "uncomfortable", and 0 occurrences of the other words.

Step 3: Term Frequency-Inverse Document Frequency (TF-IDF)

Let's calculate TF-IDF weights to give more importance to distinctive words.

Assume we have a corpus of 1000 movie reviews with these document frequencies:

Word	**# of documents containing word**	**IDF**
movie	980	log(1000/980) = 0.02
great	400	log(1000/400) = 0.92
theater	300	log(1000/300) = 1.20
uncomfortable	50	log(1000/50) = 3.00

TF-IDF vector:

[1×0.02, 1×0.92, 1×1.20, 1×3.00, 0, 0, 0, 0] = [0.02, 0.92, 1.20, 3.00, 0, 0, 0, 0]

Notice how "uncomfortable" has a much higher weight than "movie" because it's rarer and more informative.

Step 4: Pre-trained Word Embeddings

Instead of sparse vectors, we can use dense word embeddings. Let's use simplified 4-dimensional embeddings:

Word	**Embedding Vector**
movie	[0.2, 0.1, -0.3, 0.4]
great	[0.6, 0.8, 0.2, 0.1]
theater	[0.3, -0.2, 0.4, 0.5]
uncomfortable	[-0.7, -0.5, -0.3, -0.2]

To get a sentence representation, we average these vectors:

([0.2, 0.1, -0.3, 0.4] + [0.6, 0.8, 0.2, 0.1] + [0.3, -0.2, 0.4, 0.5] + [-0.7, -0.5, -0.3, -0.2])/4 = [0.1, 0.05, 0.0, 0.2]

Step 5: Sentiment Lexicon Approach

Let's use a lexicon with sentiment scores:

Word	**Sentiment Score**
movie	0 (neutral)
great	+3 (positive)
theater	0 (neutral)
uncomfortable	-2 (negative)

Computing overall sentiment:

0 + 3 + 0 + (-2) = +1

This gives a slightly positive sentiment overall.

Step 6: Machine Learning Classification

Now let's use our feature vector with a logistic regression model:

For our TF-IDF features: [0.02, 0.92, 1.20, 3.00, 0, 0, 0, 0]

Assume we have a trained logistic regression model with weights:

Weights = [-0.1, 2.5, 0.0, -1.8, 2.0, -2.2, 1.5, -1.7]

Bias = 0.5

Calculate the dot product plus bias:

(0.02 × -0.1) + (0.92 × 2.5) + (1.20 × 0.0) + (3.00 × -1.8) + ... + 0.5

= -0.002 + 2.3 + 0 - 5.4 + 0.5

= -2.602

Apply the logistic function to get probability:

P(positive) = 1/(1 + e^2.602) = 1/14.5 = 0.069 (6.9%)

Therefore, P(negative) = 93.1%

The model classifies this as negative, emphasizing "uncomfortable" despite "great" being present.

Step 7: Deep Learning Approach

A deep learning model processes the sequence using mechanisms like attention:

1. Create contextualized embeddings for each word
2. Attention mechanism might assign these weights to words:
    - movie: 0.12
    - great: 0.29
    - theater: 0.18
    - uncomfortable: 0.41
3. The attention-weighted representation emphasizes "uncomfortable" but still considers "great"
4. Final layer applies a classifier:

Positive score: 0.48

Negative score: 0.52

The deep learning model concludes this is slightly negative sentiment (52%).

Step 8: Aspect-Based Sentiment Analysis

We can break down sentiment by aspects:

- Movie aspect: +3 (positive)
- Theater aspect: -2 (negative)

This reveals that the mixed sentiment is because there are two different aspects being discussed.

Example 1: Sentiment Analysis of a Student's Feedback on a Class

Text: "The lesson was fun, but the homework was too hard."

Step 1: Preprocessing and Tokenization

- **Explanation:** We break the sentence into individual words (tokens) to analyze them.
- **Process:** Split the sentence into tokens:
  ["The", "lesson", "was", "fun", ",", "but", "the", "homework", "was", "too", "hard", "."]
- **Simplify:** Remove punctuation (,, .) and stopwords ("the", "was", "but", "too")—words that don't carry much meaning.
  Resulting Tokens: ["lesson", "fun", "homework", "hard"]

Step 2: Feature Extraction - Bag of Words (BoW)

- **Explanation:** Create a list of words (vocabulary) and count how many times each appears in the sentence.
- **Vocabulary:** Assume this classroom vocabulary:
  ["lesson", "fun", "homework", "hard", "boring", "easy", "great", "tough"]
  Indices: [0, 1, 2, 3, 4, 5, 6, 7]
- **BoW Vector:** Count occurrences in the sentence:
  [1, 1, 1, 1, 0, 0, 0, 0]
  (1 "lesson", 1 "fun", 1 "homework", 1 "hard", 0 of the others)

Step 3: Term Frequency-Inverse Document Frequency (TF-IDF)

- **Explanation:** Give more weight to rare, important words. Common words get less weight.
- **Assume Corpus:** 100 classroom reviews with these document frequencies:
  - "lesson": 90 reviews → IDF = log(100/90) = ?
  - "fun": 30 reviews → IDF = log(100/30) = ?
  - "homework": 80 reviews → IDF = log(100/80) = ?
  - "hard": 20 reviews → IDF = log(100/20) = ?

IDF = log(N / df)

Where:

- N = Total number of documents in the corpus.
- df = Number of documents containing the word.
- log = Logarithm (typically base 10 in NLP, unless specified otherwise).

1. "lesson"

DF = $\log_{10}$(100 / 90) = $\log_{10}$(1.1111)

Compute:

- $\log_{10}(1.1111) \approx 0.0458$ (using a calculator or math library).

Correct IDF ≈ 0.0458 (rounded to 0.05 for simplicity in some contexts).

Original: 0.11 → This seems off. Let's check why later.

2. "fun"

- **df = 30**
- **N / df = 100 / 30 = 3.3333...**
- **IDF = $\log_{10}$(100 / 30) = $\log_{10}$(3.3333)**
- **Compute:**
  - $\log_{10}(3.3333) \approx 0.5229$
- **Correct IDF ≈ 0.523** (rounded).
- **Original: 1.20** → This is incorrect; it's too high.

3. "homework"

- **df = 80**
- **N / df = 100 / 80 = 1.25**
- **IDF = $\log_{10}$(100 / 80) = $\log_{10}$(1.25)**
- **Compute:**
  - $\log_{10}(1.25) \approx 0.0969$
- **Correct IDF ≈ 0.097** (rounded).
- **Original: 0.22** → Also wrong.

4. "hard"

- **df = 20**
- **N / df = 100 / 20 = 5**
- **IDF = $\log_{10}$(100 / 20) = $\log_{10}$(5)**
- **Compute:**
  - $\log_{10}(5) \approx 0.6990$
- **Correct IDF ≈ 0.699** (rounded).
- **Original: 1.61** → Incorrect.

Corrected IDF Values

Using $\log_{10}$ (base 10):

- "lesson": $\log_{10}(100/90) = \log_{10}(1.1111) \approx 0.0458$
- "fun": $\log_{10}(100/30) = \log_{10}(3.3333) \approx 0.5229$
- "homework": $\log_{10}(100/80) = \log_{10}(1.25) \approx 0.0969$
- "hard": $\log_{10}(100/20) = \log_{10}(5) \approx 0.6990$

Rounded for simplicity:

- "lesson": **0.046**
- "fun": **0.523**
- "homework": **0.097**
- "hard": **0.699**

- **TF-IDF Vector:** Multiply BoW counts by IDF:
- **IDF Vector:** [0.0458, 0.5229, 0.0969, 0.6990, 0, 0, 0, 0] (0 for words not in the sentence).

Using $\log_{10}$ IDF

- BoW: [1, 1, 1, 1, 0, 0, 0, 0]
- IDF: [0.0458, 0.5229, 0.0969, 0.6990, 0, 0, 0, 0]
- TF-IDF:
  - "lesson": 1 × 0.0458 = 0.0458
  - "fun": 1 × 0.5229 = 0.5229
  - "homework": 1 × 0.0969 = 0.0969
  - "hard": 1 × 0.6990 = 0.6990
  - Others: 0 × 0 = 0
- **TF-IDF Vector:** [0.0458, 0.5229, 0.0969, 0.6990, 0, 0, 0, 0]

Insight: "hard" and "fun" stand out more than "lesson" or "homework" because they're less common.

To determine the **sentiment** using the **TF-IDF vector** [0.0458, 0.5229, 0.0969, 0.6990, 0, 0, 0, 0], we need to connect these numerical weights to a sentiment prediction (e.g., positive, negative, or neutral). The TF-IDF vector alone doesn't directly give sentiment—it's a feature representation of the text. We need an additional step, like a lexicon-based approach or a machine learning classifier, to interpret it. Since the insight highlights that "hard" and "fun" stand out due to their rarity, let's explore two practical ways to derive sentiment from this vector, focusing on the example sentence: "The lesson was fun, but the homework was too hard."
Meaning: • "lesson": 0.0458 (common, low weight) • "fun": 0.5229 (rarer, higher weight) • "homework": 0.0969 (common, low weight) • "hard": 0.6990 (rarest, highest weight) **Insight:** "hard" (0.6990) and "fun" (0.5229) dominate because they're less common in the corpus, making them more distinctive. **Insight** • "hard" has a bigger impact because it's rarer (higher IDF), aligning with the observation that it stands out more than "lesson" or "homework."

Step 4: Sentiment Lexicon Approach

- **Explanation:** Use a dictionary with sentiment scores to judge each word.
- **Lexicon:**

Match each word to its sentiment score:

- "lesson" → 0 (neutral, no strong feeling)
- "fun" → +2 (positive, makes you feel good)
- "homework" → 0 (neutral, just a thing, no emotion)
- "hard" → -2 (negative, suggests difficulty or displeasure)
- **Calculate Sentiment:** Add the scores:
  0 + 2 + 0 + (-2) = 0

Multiply TF-IDF by Sentiment Scores

Instead of just adding lexicon scores (like in the basic lexicon approach), we multiply each word's TF-IDF value by its sentiment score to reflect its importance:

- "lesson": 0.0458 × 0 = 0 (neutral, no contribution)
- "fun": 0.5229 × 2 = 1.0458 (positive contribution)
- "homework": 0.0969 × 0 = 0 (neutral, no contribution)
- "hard": 0.6990 × -2 = -1.3980 (negative contribution)
- Others: 0 × anything = 0

Sum the Weighted Scores

- Total = 0 + 1.0458 + 0 + (-1.3980)
  = 1.0458 - 1.3980 = -0.3522

Interpret the Result

- **Score:** -0.3522
- **Rule:**
  - Positive (> 0) = Positive sentiment
  - Negative (< 0) = Negative sentiment
  - Zero (≈ 0) = Neutral
- **Sentiment:** Slightly **negative**, because "hard" (-1.3980) outweighs "fun" (1.0458) due to its higher TF-IDF weight (0.6990 vs. 0.5229).

Interpret the Total Score

- **Total Score:** 0
- **What It Means:**
  - Positive numbers (e.g., > 0) = Positive sentiment.
  - Negative numbers (e.g., < 0) = Negative sentiment.
  - Zero = Neutral sentiment.
- **Result:** The sentiment of "The lesson was fun, but the homework was too hard" is **neutral**. The positive "fun" (+2) and negative "hard" (-2) balance each other out.

How to Do This in Practice

Let's say you want to try this yourself with a new sentence. Here's a simple process:

1. **Pick a Sentence:**
   Example: "The class was boring and long."
2. **Simplify It:**
   Remove stopwords ("the", "was", "and"): **"class", "boring", "long"**.
3. **Create or Use a Lexicon:**
   Let's make a small one:
   - "class": 0 (neutral)
   - "boring": -2 (negative)
   - "long": -1 (negative)
4. **Score the Words:**
   - "class": 0
   - "boring": -2
   - "long": -1
5. **Add Them Up:**
   0 + (-2) + (-1) = -3
6. **Interpret:**
   -3 is negative, so the sentiment is **negative**.

Tips for Building Your Own Lexicon

- **Start Small:** Assign scores based on your intuition (e.g., +1 to +3 for positive, -1 to -3 for negative, 0 for neutral).
- **Examples:**
  - Positive: "great" (+3), "happy" (+2), "good" (+1)
  - Negative: "awful" (-3), "bad" (-2), "tired" (-1)
  - Neutral: "book" (0), "table" (0)

Result: Neutral overall—positive and negative cancel out.

Method 2: Machine Learning Classifier

The TF-IDF vector is typically used as input to a trained machine learning model (e.g., logistic regression) to predict sentiment. Here's how that works:

Step 1: Assume a Trained Model

Let's use a simple logistic regression model with:

- **Weights:** [-0.1, 2.5, 0.0, -1.8, -2.0, 1.5, 2.0, -1.7] (one for each word in the vocabulary)
- **Bias:** 0.5 (a constant added to the result)

- These weights would come from training on labeled data (e.g., positive/negative reviews), but we'll assume them for demonstration.

Step 2: Compute the Dot Product

Multiply each TF-IDF value by its corresponding weight and sum them:

- TF-IDF: [0.0458, 0.5229, 0.0969, 0.6990, 0, 0, 0, 0]
- Weights: [-0.1, 2.5, 0.0, -1.8, -2.0, 1.5, 2.0, -1.7]
- Calculation:
  - "lesson": 0.0458 × -0.1 = -0.00458
  - "fun": 0.5229 × 2.5 = 1.30725
  - "homework": 0.0969 × 0.0 = 0
  - "hard": 0.6990 × -1.8 = -1.2582
  - Others: 0 × anything = 0
- Sum: -0.00458 + 1.30725 + 0 + (-1.2582) = -0.95553
- Add Bias: -0.95553 + 0.5 = -0.45553

Step 3: Apply Logistic Function

Convert the score to a probability of positive sentiment:

- P(positive) = 1 / (1 + e^(-score))
- Score = -0.45553
- e^(-(-0.45553)) = e^0.45553 ≈ 1.577
- P(positive) = 1 / (1 + 1.577) = 1 / 2.577 ≈ 0.388 (38.8%)
- P(negative) = 1 - 0.388 = 0.612 (61.2%)

Let's calculate e^{0.45553} step by step in a clear and practical way. This comes up in your logistic regression example when converting a score into a probability, so I'll explain it as if you're seeing it for the first time, and I'll show multiple ways to compute it—by hand (approximately), with a calculator, or programmatically.

What Is e^{0.45553} ?

- e is a special mathematical constant, approximately **2.71828**, called the base of the natural logarithm.
- e^{0.45553} means raising e to the power of 0.45553. It's the exponential function, often used in probability calculations (like the logistic function).

Method 1: Using a Calculator

The easiest way is to use a scientific calculator or phone app:

1. Enter **0.45553**.
2. Press the e^x button (sometimes labeled "exp" or " e^ ").
3. Result: $e^{0.45553} \approx 1.577$ (rounded to 3 decimal places).

Step 4: Interpret

- **Sentiment: Negative** (61.2% probability), because "hard"'s negative weight (-1.8) and high TF-IDF (0.6990) pull the score down, despite "fun"'s positive contribution.

Insight

- The model emphasizes "hard" (negative) over "fun" (positive), consistent with its higher TF-IDF weight.

Comparing the Two Methods

- **Lexicon + TF-IDF:** -0.3522 (slightly negative)
- **Logistic Regression:** 61.2% negative
- **Conclusion:** Both suggest a negative sentiment, driven by "hard"'s dominance (TF-IDF = 0.6990), though "fun" (0.5229) softens it.

How to Choose a Method?

1. **Lexicon + TF-IDF:**
    - **Pros:** Simple, no training needed, interpretable.
    - **Cons:** Ignores context (e.g., "too hard" vs. "hard work"), assumes equal sentiment strength.
    - **Use When:** You have a good lexicon and want a quick estimate.
2. **Machine Learning:**
    - **Pros:** Learns from data, handles nuance (if trained well).
    - **Cons:** Needs labeled data and a trained model.
    - **Use When:** You have training data and want accuracy.

Final Sentiment

For "The lesson was fun, but the homework was too hard":

- **Sentiment: Slightly negative**
- **Why:** "hard" (0.6990) outweighs "fun" (0.5229) in both methods, reflecting its rarity and negative connotation.

Example 2: Sentiment Analysis of a Teacher's Comment

Text: "The students were amazing, but the room was noisy."

Step 1: Preprocessing and Tokenization

- **Process:** Tokenize the sentence:
  ["The", "students", "were", "amazing", ",", "but", "the", "room", "was", "noisy", "."]
- **Simplify:** Remove punctuation and stopwords ("the", "were", "but", "was"):
  Resulting Tokens: ["students", "amazing", "room", "noisy"]

Step 2: Feature Extraction - Bag of Words (BoW)

- **Vocabulary:** ["students", "amazing", "room", "noisy", "smart", "quiet", "great", "loud"]
  Indices: [0, 1, 2, 3, 4, 5, 6, 7]
- **BoW Vector:** [1, 1, 1, 1, 0, 0, 0, 0]
  (1 "students", 1 "amazing", 1 "room", 1 "noisy")

Step 3: Term Frequency-Inverse Document Frequency (TF-IDF)

- **Assume Corpus:** 200 classroom comments:
  (N = 200).
- **Document Frequencies (df):**
  - "students": 180
  - "amazing": 40
  - "room": 150
  - "noisy": 60
- **IDF Calculation (using $\log_{10}$):**
  - "students": $\log_{10}(200/180) = \log_{10}(1.1111) \approx 0.0458$
  - "amazing": $\log_{10}(200/40) = \log_{10}(5) \approx 0.6990$
  - "room": $\log_{10}(200/150) = \log_{10}(1.3333) \approx 0.1249$
  - "noisy": $\log_{10}(200/60) = \log_{10}(3.3333) \approx 0.5229$
  - Others ("smart", "quiet", "great", "loud"): Not in sentence, so TF = 0.
- **TF-IDF Vector:** Multiply BoW (TF) by IDF:
  - "students": 1 × 0.0458 = 0.0458
  - "amazing": 1 × 0.6990 = 0.6990
  - "room": 1 × 0.1249 = 0.1249
  - "noisy": 1 × 0.5229 = 0.5229
  - Others: 0 × anything = 0
    TF-IDF Vector: [0.0458, 0.6990, 0.1249, 0.5229, 0, 0, 0, 0]
- **Insight:** "amazing" (0.6990) and "noisy" (0.5229) stand out due to rarity; "students" and "room" are common, so lower weights.

Step 4: Pre-trained Word Embeddings

- **Simplified 4D Embeddings**

- "students": [0.3, 0.2, 0.1, 0.4]
- "amazing": [0.7, 0.6, 0.2, 0.1]
- "room": [0.1, -0.1, 0.3, 0.5]
- "noisy": [-0.5, -0.4, -0.2, -0.3]

- **Sentence Representation:** Average the vectors:
  - Sum:
    - Dimension 1: 0.3 + 0.7 + 0.1 + (-0.5) = 0.6
    - Dimension 2: 0.2 + 0.6 + (-0.1) + (-0.4) = 0.3
    - Dimension 3: 0.1 + 0.2 + 0.3 + (-0.2) = 0.4
    - Dimension 4: 0.4 + 0.1 + 0.5 + (-0.3) = 0.7
    - Total: [0.6, 0.3, 0.4, 0.7]
  - Average (divide by 4): [0.6/4, 0.3/4, 0.4/4, 0.7/4] = [0.15, 0.075, 0.1, 0.175]
- **Result:** Sentence embedding = [0.15, 0.075, 0.1, 0.175]. This could be fed into a classifier.

Step 5: Sentiment Lexicon Approach

- **Lexicon (from Example 2):**
  - "students": 0 (neutral)
  - "amazing": +3 (positive)
  - "room": 0 (neutral)
  - "noisy": -1 (negative)
- **Calculate Sentiment:** Sum the scores:
  - "students": 0
  - "amazing": +3
  - "room": 0
  - "noisy": -1
  - Total: 0 + 3 + 0 + (-1) = 2
- **Result: Positive** (+2), as "amazing" (+3) outweighs "noisy" (-1).

Weighted by TF-IDF (Optional Enhancement):

- Multiply lexicon scores by TF-IDF:
  - "students": 0 × 0.0458 = 0
  - "amazing": 3 × 0.6990 = 2.0970
  - "room": 0 × 0.1249 = 0
  - "noisy": -1 × 0.5229 = -0.5229
  - Total: 0 + 2.0970 + 0 + (-0.5229) = 1.5741
- **Result:** Still **positive** (1.5741), but "amazing" dominates more due to its higher TF-IDF.

Step 6: Deep Learning Approach

- **Attention Weights (from Example 2):**
  - "students": 0.15
  - "amazing": 0.35
  - "room": 0.20
  - "noisy": 0.30
- **Weighted Representation:** Combine with embeddings and classifier (simplified):
  - Assume a classifier outputs:
    - Positive score: 0.55
    - Negative score: 0.45
- **Result: Slightly positive** (55% positive), as "amazing" (0.35) slightly edges out "noisy" (0.30).

Detailed Calculation (Hypothetical):

- Weighted embeddings (multiply each vector by attention, sum, average):
  - "students": [0.3, 0.2, 0.1, 0.4] × 0.15 = [0.045, 0.03, 0.015, 0.06]
  - "amazing": [0.7, 0.6, 0.2, 0.1] × 0.35 = [0.245, 0.21, 0.07, 0.035]
  - "room": [0.1, -0.1, 0.3, 0.5] × 0.20 = [0.02, -0.02, 0.06, 0.1]
  - "noisy": [-0.5, -0.4, -0.2, -0.3] × 0.30 = [-0.15, -0.12, -0.06, -0.09]
  - Sum: [0.16, 0.1, 0.085, 0.105]
  - Average: [0.04, 0.025, 0.02125, 0.02625]
- Classifier (assume weights [2, 3, 1, -1], bias 0.5):
  - Dot product: 0.04×2 + 0.025×3 + 0.02125×1 + 0.02625×(-1) + 0.5 ≈ 0.65
  - P(positive)=1/(1+$e^{-0.65}$)≈0.657 P(positive) = 1/(1 + e^{-0.65}) \approx 0.657 P(positive)=1/(1+e−0.65)≈0.657 (65.7%)
- **Result: Positive** (65.7%).

Step 7: Aspect-Based Sentiment Analysis

- **Aspects:**
  - **Students:** "amazing" → +3 (positive)
  - **Room:** "noisy" → -1 (negative)
- **Result:**
  - Positive about students (+3)
  - Negative about the room (-1)
- **Overall:** Mixed, but leans positive due to stronger positive sentiment.

Final Sentiment Summary

- **Lexicon:** Positive (+2)
- **TF-IDF + Lexicon:** Positive (1.5741)

- **Deep Learning:** Slightly positive (55% or 65.7%)
- **Aspect-Based:** Mixed (positive on students, negative on room)
- **Conclusion: Slightly positive overall**, driven by "amazing" outweighing "noisy" in most methods, though the room's negativity tempers it.

Insights

- "amazing" (TF-IDF 0.6990, lexicon +3) consistently pulls toward positive.
- "noisy" (TF-IDF 0.5229, lexicon -1) pulls negative but is weaker.
- The balance reflects the mixed nature of the comment.

- **TF-IDF Vector:**
  [1×0.11, 1×1.61, 1×0.29, 1×1.20, 0, 0, 0, 0] = [0.11, 1.61, 0.29, 1.20, 0, 0, 0, 0]
  Insight: "amazing" and "noisy" have higher weights because they're less common.

Step 4: Pre-trained Word Embeddings

- **Explanation:** Use dense vectors to capture word meanings, then average them for the sentence.
- **Simplified Embeddings (4D):**
  - "students": [0.3, 0.2, 0.1, 0.4]
  - "amazing": [0.7, 0.6, 0.2, 0.1]
  - "room": [0.1, -0.1, 0.3, 0.5]
  - "noisy": [-0.5, -0.4, -0.2, -0.3]

What Do These Numbers Mean?

- Each number in the vector represents some aspect of the word's meaning, learned from how the word is used in language.
- For example:
  - The first number might relate to "people vs. things" (positive for "students," negative for "noisy").
  - The second might relate to "positivity" (high for "amazing," low for "noisy").
  - The exact meaning of each dimension isn't something we define—it's learned by the model and abstract.
- **Key Point:** Words with similar meanings have similar vectors, and the differences between vectors show how words differ.

- **Sentence Vector:** Average the vectors:
  ([0.3, 0.2, 0.1, 0.4] + [0.7, 0.6, 0.2, 0.1] + [0.1, -0.1, 0.3, 0.5] + [-0.5, -0.4, -0.2, -0.3])/4 = [0.15, 0.075, 0.1, 0.175]
  Insight: This vector can be fed into a model for classification.

Step 5: Sentiment Lexicon Approach

- **Lexicon:**
  - "students": 0 (neutral)
  - "amazing": +3 (positive)
  - "room": 0 (neutral)
  - "noisy": -1 (negative)
- **Calculate Sentiment:** 0 + 3 + 0 + (-1) = +2
  Result: Slightly positive overall.

Step 6: Deep Learning Approach

- **Explanation:** Use a model with attention to focus on key words.
- **Attention Weights:**
  - "students": 0.15
  - "amazing": 0.35
  - "room": 0.20
  - "noisy": 0.30
- **Classifier Output:** Positive: 0.55, Negative: 0.45
  Result: Slightly positive, with "amazing" slightly outweighing "noisy."

Step 7: Aspect-Based Sentiment Analysis

- **Aspects:**
  - Students: "amazing" → +3 (positive)
  - Room: "noisy" → -1 (negative)
    Result: Positive about students, negative about the room.

Classroom Activity:

- **Task for Students:** Change one word (e.g., "noisy" to "quiet") and predict how the sentiment changes using the lexicon method.

Assignment Question with solution.

Question 1: Preprocessing and Tokenization

Question:

Given the sentence: "The lecture was exciting, but the room was noisy."

1. Tokenize the sentence.
2. Remove punctuation and stopwords (assume stopwords are "the", "was", "but").
3. List the remaining content words for sentiment analysis.

Solution:

Step 1: Tokenization

- Split the sentence into individual tokens (words and punctuation).

Step	Action	Result
1	Split sentence	["The", "lecture", "was", "exciting", ",", "but", "the", "room", "was", "noisy", "."]

Step 2: Remove Punctuation and Stopwords

- Remove punctuation (",", ".") and stopwords ("the", "was", "but").

Step	Action	Result
2	Remove punctuation and stopwords	["lecture", "exciting", "room", "noisy"]

Step 3: Content Words

- Identify the remaining words as content words for sentiment analysis.

Step	Action	Result
3	Identify content words	["lecture", "exciting", "room", "noisy"]

Answer:

1. Tokens: ["The", "lecture", "was", "exciting", ",", "but", "the", "room", "was", "noisy", "."]
2. After removal: ["lecture", "exciting", "room", "noisy"]
3. Content words: ["lecture", "exciting", "room", "noisy"]

Question 2: Bag of Words (BoW) Feature Extraction

Question:

For the sentence: "The movie was great, but the sound was terrible."

Assume a vocabulary: ["movie", "great", "sound", "terrible", "fun", "bad"].

1. Construct the Bag of Words (BoW) vector for the sentence.
2. Explain what the BoW vector represents.

Solution:

Step 1: Tokenize and Remove Stopwords

- Tokenize and remove punctuation and stopwords (assume "the", "was", "but").

- Tokens: ["movie", "great", "sound", "terrible"]

Step 2: Construct BoW Vector

- Count occurrences of each vocabulary word in the sentence.

Vocabulary Word	Count in Sentence	BoW Vector Position
movie	1	0
great	1	1
sound	1	2
terrible	1	3
fun	0	4
bad	0	5

- BoW Vector: [1, 1, 1, 1, 0, 0]

Step 3: Explanation

- The BoW vector represents the frequency of each vocabulary word in the sentence, ignoring order and grammar.

Answer:

1. BoW Vector: [1, 1, 1, 1, 0, 0]
2. The BoW vector shows the count of each vocabulary word present in the sentence, providing a simple numerical representation for analysis.

Question 3: TF-IDF Calculation

Question:

Given two sentences:

- Sentence 1: "The film was awesome."
- Sentence 2: "The seats were bad."

  Assume a corpus with 200 documents and document frequencies (df):
- "film": 180, "awesome": 50, "seats": 120, "bad": 80

1. Calculate the IDF for each word using $\log_{10}$.
2. Compute the TF-IDF vector for Sentence 1.
3. Explain why "awesome" has a higher IDF than "film."

Solution:

Step 1: Calculate IDF

- IDF = $\log_{10}(N / df)$, where N = 200.

Word	df	N / df	IDF = $\log_{10}(N / df)$
film	180	200/180 = 1.111	$\log_{10}(1.111) \approx 0.0458$
awesome	50	200/50 = 4	$\log_{10}(4) \approx 0.6021$
seats	120	200/120 = 1.667	$\log_{10}(1.667) \approx 0.2218$

Word	df	N / df	IDF = $\log_{10}$(N / df)
bad	80	200/80 = 2.5	$\log_{10}(2.5) \approx 0.3979$

Step 2: TF-IDF for Sentence 1

- Sentence 1: "The film was awesome" → Tokens: ["film", "awesome"] (after removing stopwords)
- TF = 1 for each word.

Word	TF	IDF	TF-IDF = TF × IDF
film	1	0.0458	0.0458
awesome	1	0.6021	0.6021
seats	0	0.2218	0
bad	0	0.3979	0

- TF-IDF Vector: [0.0458, 0.6021, 0, 0]

Step 3: Explanation

- "awesome" appears in fewer documents (df = 50) than "film" (df = 180), making it rarer and more distinctive, resulting in a higher IDF.

Answer:

1. IDF: film = 0.0458, awesome = 0.6021, seats = 0.2218, bad = 0.3979
2. TF-IDF for Sentence 1: [0.0458, 0.6021, 0, 0]
3. "awesome" has a higher IDF because it is less common across the corpus, making it more informative.

Question 4: Sentiment Lexicon Approach

Question:

For the sentence: "The lesson was dull, but the teacher was amazing."

Use the following sentiment lexicon:

- "dull": -1, "amazing": +3, "lesson": 0, "teacher": 0

1. Calculate the overall sentiment score.
2. Determine if the sentiment is positive, negative, or neutral.
3. Explain how the sentiment score is computed.

Solution:

Step 1: Identify Sentiment-Bearing Words

- Tokens: ["lesson", "dull", "teacher", "amazing"] (after removing stopwords)
- Lexicon scores: "dull" = -1, "amazing" = +3, "lesson" = 0, "teacher" = 0

Step 2: Calculate Sentiment Score

- Sum the scores of sentiment-bearing words.

Word	Score
lesson	0
dull	-1
teacher	0
amazing	+3
Total	**2**

Step 3: Determine Sentiment

- Rule: > 0 is positive, < 0 is negative, = 0 is neutral
- Total = 2 > 0 → Positive

Step 4: Explanation

- The sentiment score is the sum of lexicon scores for each word, reflecting the overall emotional tone.

Answer:

1. Sentiment score = 2
2. Positive
3. The score is computed by summing the sentiment scores of all words based on the lexicon.

Question 5: Machine Learning Classification with TF-IDF

Question:

Given the TF-IDF vector for a sentence: [0.05, 0.60, 0.10, 0.80] (for words: "fun", "tough", "class", "exam")

Assume a logistic regression model with weights: [2.5, -1.8, 0.0, -2.0] and bias = 0.3.

1. Compute the dot product of the TF-IDF vector and the weights.
2. Add the bias to the dot product.
3. Calculate the probability of positive sentiment using the logistic function.
4. Determine if the model predicts positive or negative sentiment (threshold = 0.5).

Solution:

Step 1: Compute Dot Product

- TF-IDF: [0.05, 0.60, 0.10, 0.80]
- Weights: [2.5, -1.8, 0.0, -2.0]
- Dot Product = (0.05 × 2.5) + (0.60 × -1.8) + (0.10 × 0.0) + (0.80 × -2.0)

Calculation	Result
0.05 × 2.5	0.125
0.60 × -1.8	-1.08
0.10 × 0.0	0.0

Calculation	Result
0.80 × -2.0	-1.6
Sum	**-2.555**

Step 2: Add Bias

- Bias = 0.3
- Total = -2.555 + 0.3 = -2.255

Step 3: Calculate Probability

- P(positive) = 1 / (1 + e^(-total))
- e^(-(-2.255)) = e^(2.255) ≈ 9.535
- P(positive) = 1 / (1 + 9.535) ≈ 1 / 10.535 ≈ 0.095 (9.5%)

Step 4: Determine Sentiment

- P(positive) = 0.095 < 0.5 → Negative

Answer:

1. Dot product = -2.555
2. Total with bias = -2.255
3. P(positive) ≈ 0.095
4. Negative sentiment