Module 5: Applied Low-Tech NLP

Rules-Based Sentiment Analysis

- **Rules-based sentiment analysis** is an approach that uses manually defined linguistic rules, lexicons, and heuristics (rather than ML) to determine whether a text expresses a positive, negative, or neutral sentiment.
- Instead of training on large datasets, we encode human knowledge into:
  - Sentiment lexicons (dictionaries of words with positive/negative scores).
  - Rules about negation, intensifiers, and context.
  - Simple scoring functions.

The Core Components of Rules-Based Sentiment Analysis

- A list of words associated with sentiment polarity (positive/negative).
- A scoring mechanism that sums up the polarity scores of words in a text. Mathematically, we sum up all the sentiments of the words w in the particular token T

$$\boxed{\text{Sentiment Score} = S = \sum_{w \in T} S(w)}$$

- **Negation Handling:** Rules to flip polarity when negation words appear. Example: If the word "not" precedes a positive word, multiply score by -1.
- **Intensifiers and Diminishers:** Words that amplify (very, extremely) or weaken (slightly, somewhat) sentiment. Example: "Very good" gets a score of $+4$ as opposed to $+2$.

The Pros and Cons of Rules-Based Sentiment Analysis

- **Advantages**
  - Transparent (easy to explain why a decision was made).
  - Works well with small datasets (no training required).
  - Easy to customize for specific industries.
- **Disadvantages**
  - Struggles with sarcasm ("Great service... not!").
  - Hard to scale to very large vocabularies.
  - Needs manual updates to lexicons and rules.

> Python libraries like VADER (**V**alence **A**ware **D**ictionary for s**E**ntiment **R**easoning) implement these rules and are widely used for social media, reviews, and business analytics.

Lexicons

- A **lexicon** is a structured list of words used by NLP systems to understand and process text.
- The contents of a lexicon are:
  - Words or phrases (unigrams, bigrams, etc.)
  - Part-of-speech tags (nouns, verbs, adjectives, etc.)
  - Morphological information (plurals, tenses, lemmas, etc.)
  - Semantic information (meanings, synonyms, antonyms, etc.)
  - Sentiment scores (positive, negative, neutral)

Types of Lexicons

- **General-Purpose Lexicons:** WordNet, Oxford Dictionary, etc.
- **Domain-Specific Lexicons:** Financial, medical, or business lexicons tailored for specialized vocabulary.
- **Sentiment Lexicons:** Lists of words with associated sentiment polarity. For example, "happy" $\implies$ positive, "bad" $\implies$ negative.

Applications of Lexicons in Business

- Using a sentiment lexicon, analyze customer reviews or Tweets. For example, the statement: "The delivery was fast but the product is terrible", can be used to match words with positive/negative lexicon scores.
- Used for keyword matching with a focus on business-specific lexicon to detect mentions of products, services, or competitors.
- Used to assign categories to emails, documents, or complaints based on the presence of lexicon terms.

A Real-World Application of Lexicons: Customer Review Analysis

Scenario: Data Scientists at your organization construct the following sentiment lexicon:

Word	**Sentiment:** S
"excellent"	$+1$
"good"	$+1$
"bad"	-1
"terrible"	$+1$
"fast"	$+1$

Task: If a customer posted: "The delivery was fast but the product is terrible", what can be concluded about this review?

- The total sentiment is $S_{\text{tot}} = S(\text{"fast"}) + S(\text{"terrible"}) = (+1) + (-1) = 0$. Thus, we conclude that we have a mixed review.

Bag of Words (BoW) Representation

- The **Bag of Words** (BoW) model is a text representation technique where a piece of text (sentence, paragraph, document) is represented as a collection of its words, ignoring grammar, word order, and context, but keeping word frequency.
- It is one of the most fundamental feature extraction methods in NLP.
- Let $V = \{w_1, w_2, \ldots, w_n\}$ be a vocabulary of size n. Suppose we want to document a sequence of d words. Then, the representation vector is

$$\boxed{\mathbf{x}_d = [f(w_1, d), f(w_2, d), \ldots, f(w_n, d)]}$$

where $f(w_i, d)$ is the frequency of word w_i in document d.

Example of BoW

Scenario : Suppose that your company wants to analyze customer reviews.

- **Review 1:** "The product is excellent".
- **Review 2:** "The service is excellent".

The vocabulary is $V = \{\text{the, product, is, excellent, service}\}$.
Task: Construct the representation vectors.

The representation vectors are:

$$\mathbf{x}_{d_1} = [1, 1, 1, 1, 0], \quad \mathbf{x}_{d_2} = [1, 0, 1, 1, 1].$$

Now these vectors can be fed into ML models for sentiment analysis, clustering, or recommendation systems.

Pros and Cons of BoW

- **Advantages**
  - Simple and fast to implement.
  - Good for traditional ML algorithms (Naïve Bayes, SVM, Logistic Regression).
- **Disadvantages**
  - Ignores word order. For example, “not good” and “good not” look the same.
  - High dimensionality, large vocabulary implies long vectors.
  - No semantic understanding. For example, “excellent” $\neq$ “great”.

> BoW is like a shopping list of words – It only cares about what words are present and how many times, not how they are arranged.

Cosine Similarity

- **Cosine similarity** is a metric used to measure how similar two vectors are, by calculating the cosine of the angle between them.
- In NLP, once text is represented as vectors (via Bag of Words, TF-IDF, or embeddings), cosine similarity tells us how close in meaning two texts are.
- Mathematically, given two vectors $\mathbf{A}$ and $\mathbf{B}$, the cosine similarity between them is given by

$$\boxed{\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} = \frac{\sum_i A_i B_i}{\left(\sqrt{\sum_i A_i^2}\right)\left(\sqrt{\sum_i B_i^2}\right)}}$$

- Interpretation:
  - If $\cos\theta = 1$, you have identical direction (perfect similarity).
  - If $\cos\theta = 0$, you have orthogonality (no similarity).
  - If $\cos\theta = -1$, you have opposite directionality (rare in text, since frequencies are non-negative).

Example of Cosine Similarity

Given the corpus composed of two documents:

- **Document 1:** "I love data science".
- **Document 2:** "I love machine learning".

The vocabulary is

$$V = \{\text{I, love, data, science, machine, learning}\}.$$

Task: Calculate the cosine similarity between the two documents.

- The representation vectors for each document are

$$\mathbf{A} = [1, 1, 1, 1, 0, 0], \quad \mathbf{B} = [1, 1, 0, 0, 1, 1].$$

- The numerator in the cosine similarity equation is

$$\mathbf{A} \cdot \mathbf{B} = (1)(1) + (1)(1) + (1)(0) + (1)(0) + (0)(1) + (0)(1) = 2.$$

- The magnitudes of the representation vectors are

$$||\mathbf{A}|| = \sqrt{(1)^2 + (1)^2 + (1)^2 + (1)^2 + (0)^2 + (0)^2} = 2,$$
$$||\mathbf{B}|| = \sqrt{(1)^2 + (1)^2 + (0)^2 + (0)^2 + (1)^2 + (1)^2} = 2.$$

- Thus, the cosine similarity is

$$\cos\theta = \frac{2}{2 \times 2} = 0.5.$$

- This means that document 1 and document 2 are 50% similar.

Why is Cosine Similarity Used in NLP?

- Unlike Euclidean distance, cosine similarity is scale-invariant (it does not matter if one document is much longer than another).
- It is excellent for comparing text in:
  - Document clustering (e.g., grouping news articles).
  - Recommendation systems (e.g., finding similar products from descriptions).
  - Plagiarism detection.
  - Semantic search engines.

(TF-IDF)

- **TF-IDF** is a numerical statistic used to reflect how important a word is in a document relative to a collection of documents (corpus).
- **TF (Term Frequency):** Measures how often a word appears in a document.
- **IDF (Inverse Document Frequency):** Reduces the weight of common words and increases the weight of rare but important words.
- Together, TF-IDF balances local importance (within a document) and global importance (across the corpus).

How is TF-IDF Calculated?

- Recall, the term frequency (TF) was calculated as

$$\boxed{TF(t,d) = \frac{f_{t,d}}{\sum_k f_{k,d}}}$$

  where $f_{t,d}$ is the raw count of term t in document d, and the denominator captures the total number of terms in document d.

- The inverse document frequency (IDF) term is calculated as

$$\boxed{IDF(t,D) = \log\left(\frac{N}{1 + |\{d \in D | t \in d\}|}\right)}$$

  where N is the total number of documents, and the denominator contains the number of documents containing t terms.

- Together, these form the TF-IDF, which is given by

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

Example TF-IDF Calculation

Scenario: Suppose that the corpus is composed of three documents:

- **Document 1:** "I love data science".
- **Document 2:** "I love machine learning".
- **Document 3:** "Data science and machine learning'.

Task: Calculate TF-IDF for the word "data" in document 1.

- The TF is

$$TF(\text{“data”, document 1}) = \frac{1}{4} = 0.25.$$

- Now, $N = 3$ (3 documents), and so we have that

$$IDF(\text{“data”}) = \log\left(\frac{3}{1+1}\right) = \log(3/2) \approx 0.4055.$$

- Thus, the TF-IDF score is

$$TF - IDF = 0.25 \times 0.4055 = 0.1014.$$

Why Use TF-IDF?

- Solves Bag of Words issue: Instead of treating all words equally, it gives more importance to unique terms.
- Removes bias of frequent words: Words like “the”, “is”, “and” do not dominate.
- Helps in search/recommendations: Matches rare but meaningful words better.

Lab 5

To apply the concepts learned in Applied Low-Tech NLP to solve real-world business problems.