SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY

(An Autonomous Institution Approved by UGC)

Yamnampet, Ghatkesar, R.R. District -501 301

Department of

COMPUTER SCIENCE and ENGINEERING

A Major Project Code Implementation seminar

on

LEXICON BASED SENTIMENT ANALYSIS

by

K. Giridhar 21311A0513

P. Rajeswari 21311A0508

J. Aaraadhya 21311A0538

Internal Guide	**Project Coordinator**	**Head of the Department**
Dr.E. Madhukar	Mrs.B.Vasundhara Devi	Dr. Aruna Varanasi
Professor	Assistant Professor	Professor

CONTENTS

ABSTRACT

In recent years, it is seen that the opinion-based postings in social media are helping to reshape business and public sentiments, and emotions have an impact on our social and political systems. Opinions are central to mostly all human activities as they are the key influencers of our behavior. Whenever we need to make a decision, we generally want to know other's opinion. Every organization and business always wants to find customer or public opinion about their products and services. Thus, it is necessary to grab and study the opinions on the Web. However, finding and monitoring sites on the web and distilling the reviews remains a big task because each site typically contains a huge volume of opinion text and the average human reader will have difficulty in identifying the polarity of each review and summarizing the opinions in them. Hence, it needs the automated sentiment analysis to find the polarity score and classify the reviews as positive or negative.

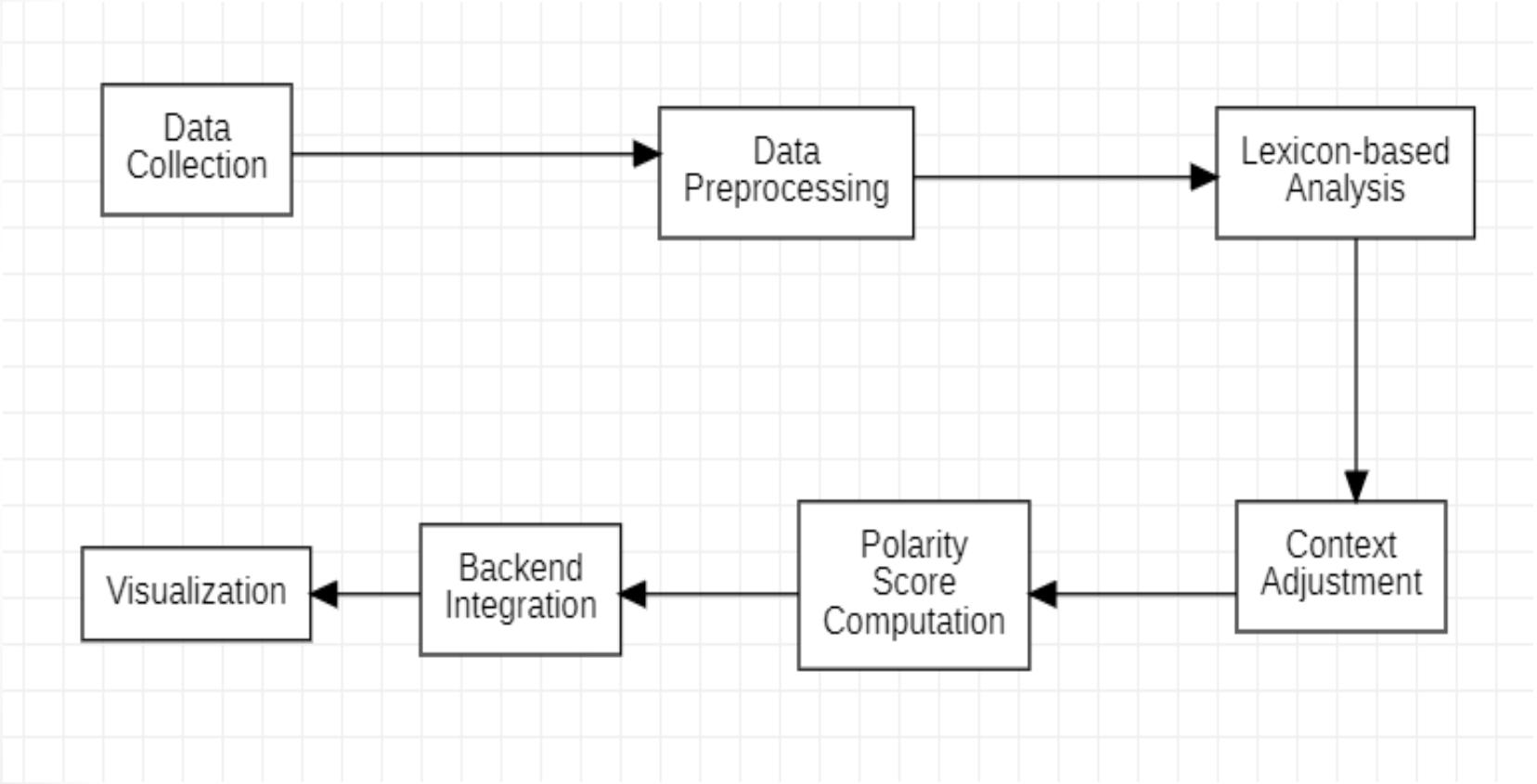
INTRODUCTION

Sentiment analysis is the process of computationally identifying and categorizing the opinions expressed in a piece of text, especially in order to determine the writer's attitude towards a particular topic, product, etc. is positive, negative or neutral. he attitude may be as one of the following scenarios. His or her judgment or evaluation Affective state i.e., the emotional state of the author when writing a review. Sentiment analysis can be highly useful in several cases. The best example is the marketing methodology. Marketing teams can use sentiment analysis to launch a new product or to determine the existing product popularity and preference. Reviews from social media can be gathered and used to assess how good or bad a product or service is doing based on customer response.

EXISTING SYSTEM

- **Limited Emotion Detection**
- **Simple Negation Handling**
- **Focuses mainly on individual words or phrases, often ignoring the broader context.**
- **Uses general-purpose models and sentiment dictionaries, often yielding suboptimal results in specialized fields (e.g., finance, healthcare, legal).**

PROPOSED SYSTEM



CODE IMPLEMENTATION

```
# ============================================================================
# class Sentiment
# ============================================================================

class Sentiment:

    # input doc and return polarity score [-1, +1]
    def get_polarity_score(input_doc, subjectivity = False, verbose = False):

        global v
        if verbose is True:
            v = True
        else:
            v = False

        if v: print("----- Input doc -----")
        if v: print(input_doc)

        raw_subsentence_pos_polarity_scores = []
        raw_subsentence_neg_polarity_scores = []

        raw_doc_pos_polarity_score = 0.0
        raw_doc_neg_polarity_score = 0.0

        raw_doc_polarity_score = 0.0

        final_doc_polarity_score = 0.0

        word_count = 0
        pos_matches = 0
        neg_matches = 0

        # get sentences
        sentences = get_sentences(input_doc)
```

```
# get subsentences for each sentence
for sentence in sentences:
    subsentences = get_subsentences(sentence)
    for subsentence in subsentences:
        if v: print("\n----- Subsentence -----")
        if v: print(subsentence)
        raw_subsentence_polarity_score = 0.0

        # check for irrealis marker and bypass current subsentence if True
        irrealis_marker = check_irrealis_markers(subsentence)
        if irrealis_marker == True:
            continue

        # check for idioms and update pos and neg polarity scores
        pos_idiom_score, neg_idiom_score, subsentence = check_for_idiom(subsentence)
        raw_subsentence_polarity_score += (pos_idiom_score + neg_idiom_score)
        # assign 1 match for each idiom
        pos_matches += pos_idiom_score
        neg_matches -= neg_idiom_score
        # assume wordcount is 1 for each idiom detected
        word_count += (pos_idiom_score - neg_idiom_score)

        # check multi exclamation and get score (coefficient) 1.1 - 2.0 depending on num of '!'
        exclamation_score = check_emphatic_multi_exclamation(subsentence)

        # check for emoticons and return emoticon score
        emoticon_score, pos_emoticon_count, neg_emoticon_count = check_emoticons(subsentence)
        pos_matches += pos_emoticon_count
        neg_matches += neg_emoticon_count
        word_count += pos_emoticon_count + neg_emoticon_count

        # check for emojis and return emoji score
        emoji_score, pos_emoji_count, neg_emoji_count = check_emojis(subsentence)
        pos_matches += pos_emoji_count
        neg_matches += neg_emoji_count
```

```
    ######################################
    # handle pos word matches            #
    ######################################

# derive word_polarity from lexica
if word in pos_lex:
    word_polarity = float(pos_lex[word])

    # if uppercasing is True, boost word_polarity
    if emphatic_uppercasing == True:
        word_polarity = word_polarity * 1.5
        if v: print("-> uppercasing detected on {}. Word polarity boosted to +{}.".format(word.upper(), word_polarity))
    # if lengthening is True, boost word_polarity
    if emphatic_lengthening == True:
        word_polarity = word_polarity * 1.5
        if v: print("-> lengthening detected on {}. Word normalized to {}. Word polarity boosted to +{}. ".format(lengthened_word, word, word_polarity))

    # get exclamation score and boost word_polarity
    if exclamation_score is None:
        exclamation_score = 1
    if exclamation_score > 1:
        word_polarity = word_polarity * exclamation_score
        if v: print("-> {} exclamations detected near {}. Word polarity boosted to +{}.".format(exclamation_count, word, round(word_polarity, 3)))
```

```
# handle negation+intensifier+pos word combo
if i > 0 and tokens[i-2].lower() in negators and tokens[i-1].lower() in intensifiers:
    intensifier = tokens[i-1]
    intensification = float(intensifiers[intensifier.lower()])
    # if uppercased, increa
    if check_emphatic_upper
        intensification += intensification * 0.1
    word_polarity = intensification * word_polarity
    # if score is low, invert polarity
    if word_polarity <= 0.5:
        word_polarity = -(word_polarity)
    # if score is high, invert+reduce polarity
    elif word_polarity > 0.5:
        word_polarity = -(word_polarity * 0.5)
    if v: print("-> neg+int+pos match: {} {} {} [{}]".format(tokens[i-2], intensifier, word, word_polarity))

# handle intensifier on pos word
elif i > 0 and tokens[i-1].lower() in intensifiers:
    intensifier = tokens[i-1]
    intensification = float(intensifiers[intensifier.lower()])
    # if uppercased, increase coefficient value by 10%
    if check_emphatic_uppercasing(intensifier) == True:
        intensification += intensification * 0.1
    word_polarity = intensification * word_polarity
    if v: print("-> int+pos match: {} {} [+{}]".format(intensifier, word, word_polarity))

# handle diminisher on pos word
elif i > 0 and tokens[i-1].lower() in diminishers:
    diminisher = tokens[i-1]
    diminishment = float(diminishers[diminisher.lower()])
    # if uppercased, reduce coefficient value by 10%
    if check_emphatic_uppercasing(diminisher) == True:
        diminishment -= diminishment * 0.1
    word_polarity = diminishment * word_polarity
    if v: print("-> dim+pos match: {} {}  [+{}]".format(diminisher, word, word_polarity))
```

(variable) intensification: float

```
    ##########################################
    # handle neg word matches                #
    ##########################################

# derive word_polarity from lexica
if word in neg_lex:
    word_polarity = float(neg_lex[word]) # assign word polarity
    # if uppercasing is True, boost word_polarity
    if emphatic_uppercasing == True:
        word_polarity = word_polarity * 1.5
        if v: print("-> uppercasing detected on {}. Word polarity boosted to {}.".format(word.upper(), word_polarity))
    # if lengthening is True, boost word_polarity
    if emphatic_lengthening == True:
        word_polarity = word_polarity * 1.5
        if v: print("-> lengthening detected on {}. Word normalized to {}. Word polarity boosted to {}. ".format(lengthened_word, word, word_polarity))

    # get exclamation score and boost word_polarity
    if exclamation_score is None:
        exclamation_score = 1
    if exclamation_score > 1:
        word_polarity = word_polarity * exclamation_score
        if v: print("-> {} exclamations detected near {}. Word polarity boosted to {}.".format(exclamation_count, word, round(word_polarity, 3)))

    # handle negation+intensifier+neg word combo
    if i > 0 and tokens[i-2].lower() in negators and tokens[i-1].lower() in intensifiers:
        intensifier = tokens[i-1]
        intensification = float(intensifiers[intensifier.lower()])
        # if uppercased, increase coefficient value by 10%
        if check_emphatic_uppercasing(intensifier) == True:
            intensification += intensification * 0.1
        word_polarity = intensification * word_polarity
        # if score is low, invert polarity
        if word_polarity >= -0.5:
            word_polarity = -(word_polarity)
        # if score is high, invert+reduce polarity
        elif word_polarity < -0.5:
            word_polarity = -(word_polarity * 0.5)
        if v: print("-> neg+int+neg match: {} {} {} [+{}]".format(tokens[i-2], intensifier, word, word_polarity))
```

```
# handle intensifier on neg word
elif i > 0 and tokens[i-1].lower() in intensifiers:
    intensifier = tokens[i-1]
    intensification = float(intensifiers[intensifier.lower()])
    # if uppercased, increase coefficient value by 10%
    if check_emphatic_uppercasing(intensifier) == True:
        intensification += intensification * 0.1
    word_polarity = intensification * word_polarity
    if v: print("-> int+neg match: {} {} [{}]".format(intensifier, word, word_polarity))

# handle diminisher on neg word
elif i > 0 and tokens[i-1].lower() in diminishers:
    diminisher = tokens[i-1]
    diminishment = float(diminishers[diminisher.lower()])
    # if uppercased, reduce coefficient value by 10%
    if check_emphatic_uppercasing(diminisher) == True:
        diminishment -= diminishment * 0.1
    word_polarity = diminishment * word_polarity
    if v: print("-> dim+neg match: {} {}  [{}]".format(diminisher, word, word_polarity))

# handle adjacent neg word
elif i > 0 and tokens[i-1].lower() in neg_lex:
    word_polarity = word_polarity * 1.5
    if v: print("-> dbl neg match: {} {} [{}]".format(tokens[i-1], word, word_polarity))

# handle negation (1-4 hops away) on word
elif i > 1 and (tokens[i-1].lower() in negators or tokens[i-2].lower() in negators or tokens[i-3].lower() in negators or tokens[i-4].lower() in negators):
    # if score is low, invert polarity
    if word_polarity >= -0.5:
        word_polarity = -(word_polarity)
    # if score is high, invert+reduce
    elif word_polarity < -0.5:
        word_polarity = -(word_polarity * 0.5)
    if v: ("-> negator applied on {}: [+{}]".format(word, word_polarity))
```

```
# compute final doc polarity score, confined to -1, 0, +1
raw_doc_polarity_score = raw_doc_pos_polarity_score + raw_doc_neg_polarity_score
final_doc_polarity_score = round(( raw_doc_polarity_score / math.sqrt((raw_doc_polarity_score * raw_doc_polarity_score) + 100) ), 4)
if final_doc_polarity_score > 1.0:
    final_doc_polarity_score == 1.0
elif final_doc_polarity_score < -1.0:
    final_doc_polarity_score == -1.0

# compute pos, neg, and neutral portions of doc, all of of which sum up to 1
if subjectivity is True:
    if word_count < 1:
        word_count = 1
    pos_portion = round((pos_matches / word_count), 4)
    neg_portion = round((neg_matches / word_count), 4)
    neutral_portion = round(1 - (pos_portion + neg_portion), 4)

    # return dict: final_doc_polarity_score, pos_portion, neg_portion, neutral_portion
    polarity_score_dict = {"polarity": final_doc_polarity_score, "pos portion": pos_portion, "neg portion": neg_portion, "neutral portion": neutral_portion}

    return(polarity_score_dict)
```

```python
# ========================================================================
# main
# ========================================================================

if __name__ == "__main__":

    s = Sentiment

    text_docs = ["I am feeling 😊😀😎👍", # positive emojis
                 "I am feeling 😡😭😨💔"] #negative emojis
    for text_doc in text_docs:
        print(s.get_polarity_score(text_doc, verbose=True))
```

```
----- Input doc -----
I am feeling 😊😀😎👍

----- Subsentence -----
I am feeling 😊😀😎👍
-> pos emoji 😊 +0.6
-> pos emoji 😀 +0.9
-> pos emoji 😎 +1
-> pos emoji 👍 +1
Raw subsentence polarity score: 3.5
Raw_subsentence_pos_polarity_scores: [3.5]
Raw_subsentence_neg_polarity_scores: []
Raw_doc_pos_polarity_score 3.5
Raw_doc_neg_polarity_score 0.0
0.3304
Raw_doc_neg_polarity_score 0.0
Raw_doc_neg_polarity_score 0.0
Raw_doc_neg_polarity_score 0.0
0.3304
Raw_doc_neg_polarity_score 0.0
Raw_doc_neg_polarity_score 0.0
0.3304
```

```
----- Subsentence -----
I am feeling 😡😭😰💔
-> neg emoji 😡 -1
-> neg emoji 😭 -0.8
-> neg emoji 😰 -0.7
-> neg emoji 💔 -0.6
Raw subsentence polarity score: -3.1
Raw_subsentence_pos_polarity_scores: []
Raw_subsentence_neg_polarity_scores: [-3.1]
Raw_doc_pos_polarity_score 0.0
Raw_doc_neg_polarity_score -3.1
-0.2961
```

OUTPUT SCREENSHOTS


Context Aware Sentiment Analysis
Choose Input Type:
Single Text
Multiple Texts
Enter Text:
I am feeling
Analyze Sentiment

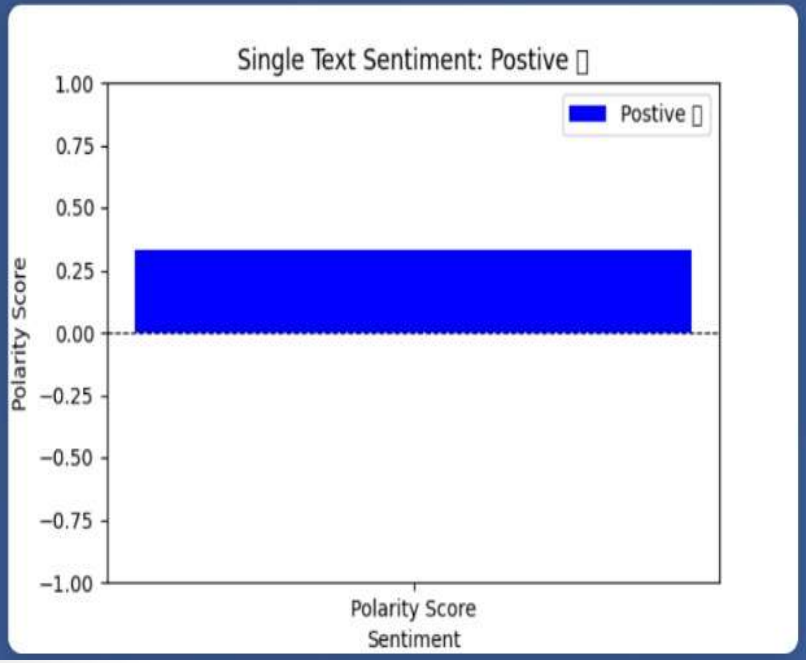
Single Text Analysis

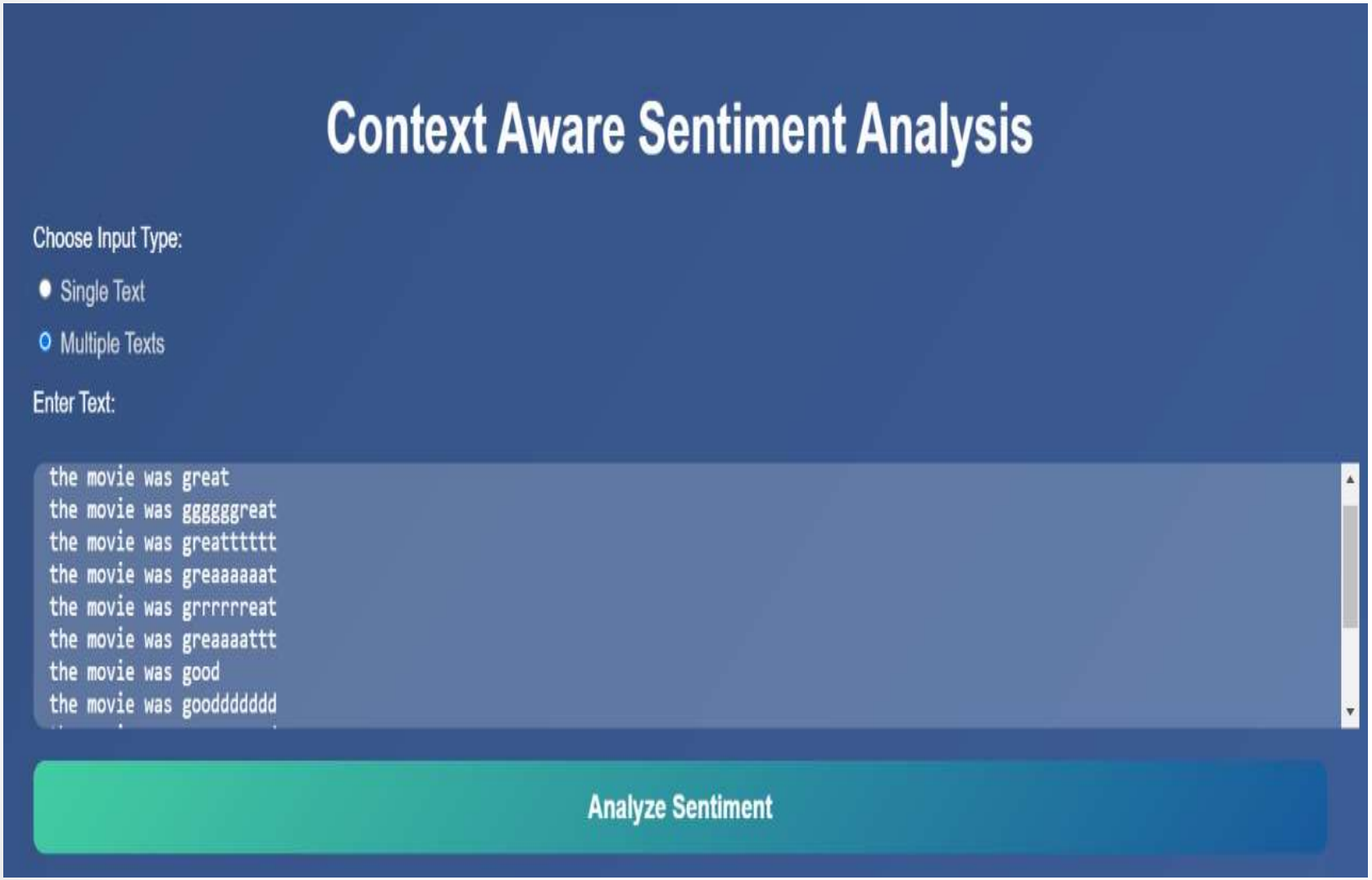
Text: I am feeling 😊😜😍👍

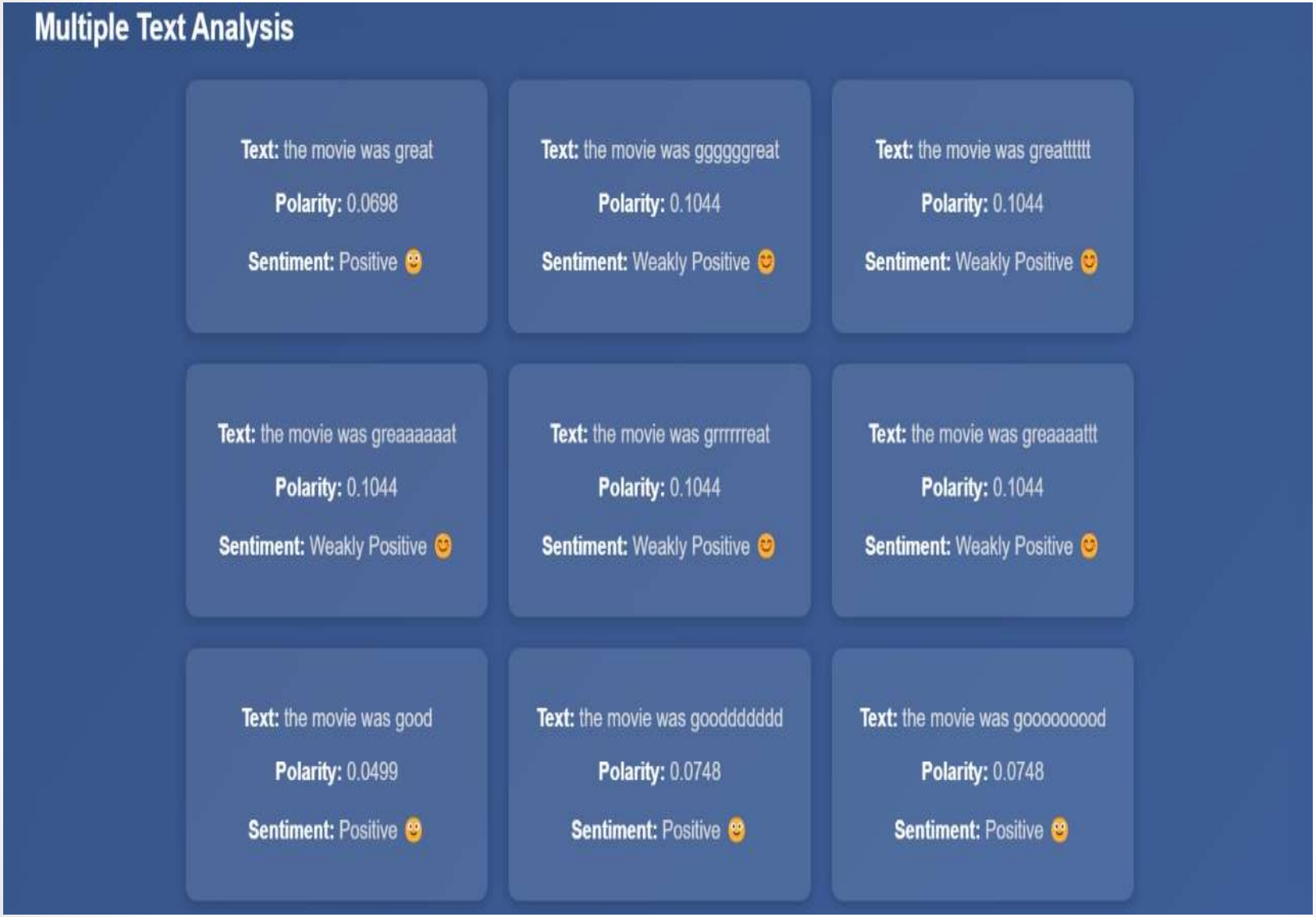
Polarity: 0.3304

Sentiment: Postive 😊

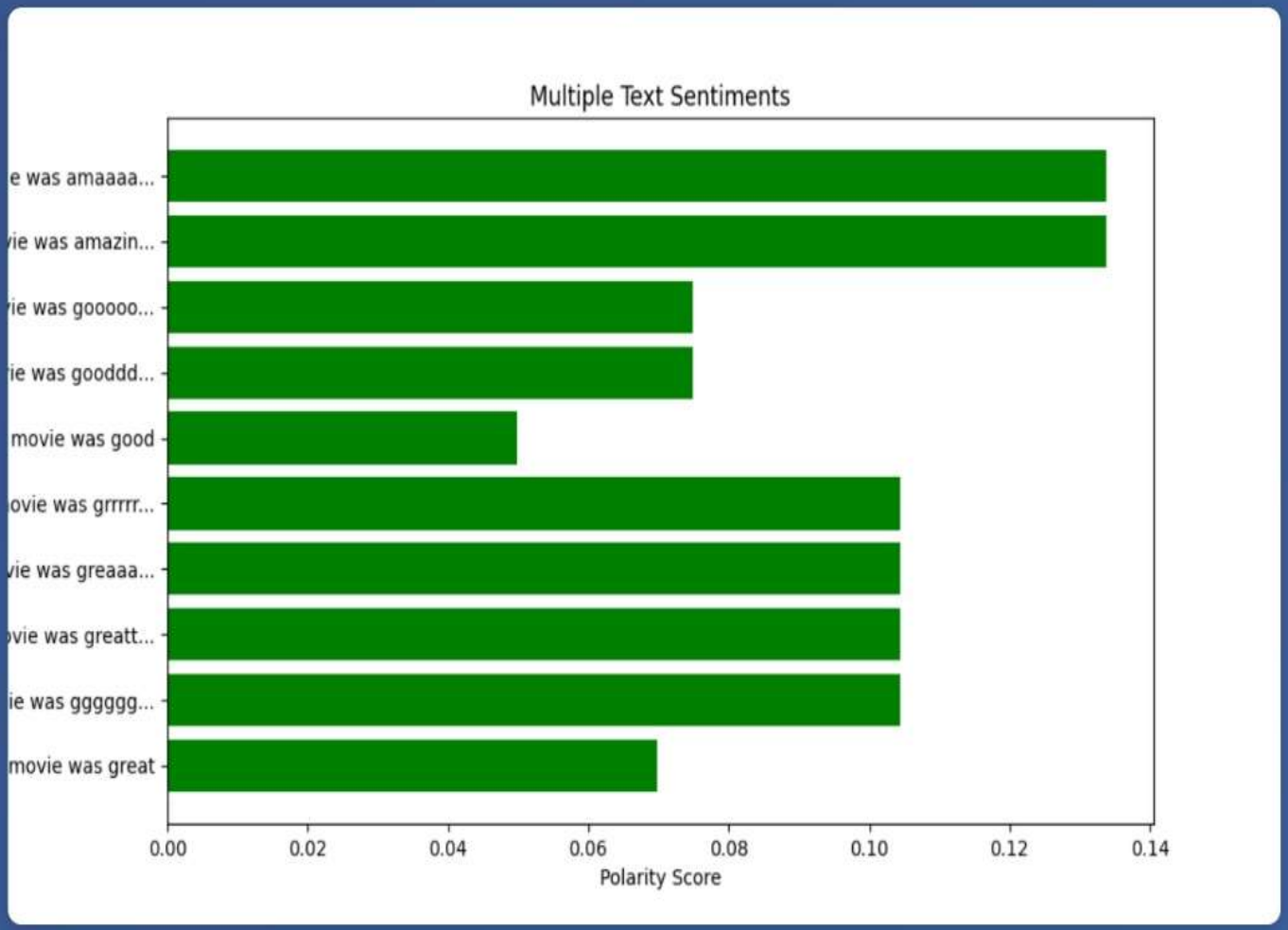
Visualization




Context Aware Sentiment Analysis
Choose Input Type:
Single Text
Multiple Texts
Enter Text:
the movie was great
the movie was ggggggreat
the movie was greatttttt
the movie was greaaaaaat
the movie was grrrrrreat
the movie was greaaaattt
the movie was good
the movie was goodddddddd
Analyze Sentiment


Multiple Text Analysis
Text: the movie was great
Polarity: 0.0698
Sentiment: Positive
Text: the movie was ggggggreat
Polarity: 0.1044
Sentiment: Weakly Positive
Text: the movie was greatttttt
Polarity: 0.1044
Sentiment: Weakly Positive
Text: the movie was greaaaaaat
Polarity: 0.1044
Sentiment: Weakly Positive
Text: the movie was grrrrrreat
Polarity: 0.1044
Sentiment: Weakly Positive
Text: the movie was greaaaattt
Polarity: 0.1044
Sentiment: Weakly Positive
Text: the movie was good
Polarity: 0.0499
Sentiment: Positive
Text: the movie was goodddddddd
Polarity: 0.0748
Sentiment: Positive
Text: the movie was gooooooood
Polarity: 0.0748
Sentiment: Positive

Visualization



REFERENCES

lexicon based sentiment analysis - Google Scholar

Lexicon-Based Methods for Sentiment Analysis | Computational Linguistics | MIT Press

A Comprehensive Study on Lexicon Based Approaches for Sentiment Analysis | Asian Journal of Computer Science and Technology

Thank You