Automatic Menu Popularity Analysis via Fuzzy Matching-Based Extraction of Customer Review

Nayla Zahira - 13523079
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: naylaazahira9@gmail.com , 13523079@std.stei.itb.ac.id

***Abstract*— Traditional menu analysis methods rely primarily on sales data, providing limited insights into customer satisfaction and preferences. To address this limitation, we developed a system that leverages fuzzy string-matching algorithms to extract menu item mentions from unstructured customer review text. The system first normalizes each review, then pinpoints dish references through a two-tier matcher that combines exact substring matching with levenshtein distance-based fuzzy matching, achieving 80% similarity threshold optimization for menu item identification. This system uses a comprehensive synonym dictionary that maps canonical menu names to their textual variations. Performance evaluation demonstrates the system can process 10^4 reviews in under 2 seconds with O(R·I·S·L) complexity, where R represents reviews, I menu items, S synonyms per item, and L average sentence length. This automated approach transforms unstructured customer feedback into actionable business intelligence, enabling restaurants to make data-driven decisions about menu optimization, item reformulation, and customer satisfaction improvements~~This electronic document is a "live" template and already defines the components of your paper [title, text, heads, etc.] in its style sheet~~.**

***Keywords*—~~component~~*fuzzy matching*; ~~*formatting*~~*menu analysis*; ~~*style*~~*customer reviews*; ~~*styling*~~*string matching*; *levenshtein distance*~~*insert (key words)*~~**

I. Introduction

In the increasingly competitive restaurant industry, understanding customer preferences and menu item popularity has become crucial for business success and strategic decision-making. Traditional methods of assessing menu performance often rely on sales data alone, which provides limited insight into the underlying reasons for customer satisfaction or dissatisfaction. With the proliferation of online review platforms such as Google Reviews, GoFood Reviews, and social media, customers now generate vast amounts of unstructured textual feedback that contains valuable information about their dining experiences and menu preferences.

The challenge of extracting meaningful insights from customer reviews lies in the inherent complexity and variability of natural language. Customers may refer to the same menu item using different names, abbreviations, or descriptions, making it difficult to automatically identify and categorize feedback related to specific dishes. For instance, a customer might refer to "Caesar salad" as "Caesar" or "the salad". This linguistic ambiguity presents a significant obstacle for automated analysis systems that rely on exact string matching

To address this problem, fuzzy matching techniques offer a promising solution by allowing for approximate string matching that can handle variations in spelling and phrasing. By leveraging algorithm such as levenshtein distance, phonetic matching, and similarity scoring, fuzzy matching can identify relationships between customer review text and menu items even when exact matches are not identified. This approach enables more comprehensive extraction of menu-related feedbacks from customer reviews, leading to more accurate menu popularity analysis.

The ability to automatically analyze menu popularity from customer reviews has significant practical implications for restaurant management. Such analysis can inform menu optimization decisions, identify underperforming items, highlight customer favorites, and provide insights into emerging food trends. Furthermore, this approach can help restaurants understand not just what customers order, but how they feel about their dining choices, enabling more targeted improvements to menu offerings and overall customer satisfaction.

This paper presents a novel approach to automatic menu popularity analysis that combines fuzzy matching algorithms with natural language processing techniques to extract and analyze customer feedback from online reviews. Our methodology addresses the challenge of mapping customer descriptions to menu items while providing quantitative insights into item popularity and customer sentiment. The proposed system demonstrates how advanced text processing techniques can transform unstructured customer feedback into actionable business intelligence for the restaurant industry.

II. Theoretical Foundation

A. Menu Popularity Analysis

Menu popularity analysis is a systematic approach to evaluating the performance and customer preference of individual menu items within a restaurant's offerings. This analysis serves as a critical component of restaurant management strategy, enabling establishments to make data-driven decisions about menu optimization, pricing, and inventory management.

Traditional menu popularity analysis primarily relies on quantitative sales data, measuring metrics such as sales volume, frequency of orders, and revenue contribution per item. However, this approach provides limited insight into the qualitative aspects of customer satisfaction and the underlying reasons for item popularity or unpopularity. ~~Hence why~~Therefore, modern approaches to menu popularity analysis incorporate multiple data sources, including customer feedback, reviews, ratings, and social media mentions, to provide a more comprehensive understanding of menu performance.

Menu popularity analysis typically involves several key components: item identification and categorization, sentiment analysis of customer feedback, popularity scoring based on mention frequency and sentiment polarity, and comparative analysis across different menu categories. The ultimate goal is to provide actionable insights that can guide menu engineering decisions, such as promoting popular items, reformulating underperforming dishes, or adjusting pricing strategies.

B. String Matching

String matching is the process of checking whether a certain pattern exists within a text. Generally, string matching algorithms have two main components:

- Text, which is a string of length n characters
- Pattern, which is a string of length m characters (where m ≤ n) that will be searched within the text.

The goal of string matching is to find all occurrences of the pattern in the text and determine the positions where the pattern is found.

This process represents one of the fundamental operations in computer science and text processing, with wide applications across various domains. In information technology, string matching is used for word or sentence searching in text editors, database search systems, and web search engines. In bioinformatics, this technique is applied for searching amino acid chains in DNA sequences and protein structure analysis. Other applications include fingerprint image matching for security systems, plagiarism detection in documents, and recommendation systems based on text similarity.

String matching is divided into two main types: literal string matching (exact matching) and fuzzy matching.

C. Exact Matching

Exact matching, also known as literal string matching, is a fundamental technique in information retrieval and text processing where two strings are considered a match only if they are identical in every character. In the context of menu popularity analysis, exact matching involves searching for customer review text that precisely corresponds to menu item names as they appear on the official menu.

Several well-established algorithms used to perform exact string matching are as followed:

- Brute Force Algorithm is the most straightforward approach that checks each position in the text to see if the pattern starts at that position [1]. The algorithm compares the pattern character by character with the text, moving one position at a time through the text when a mismatch occurs. While simple to implement and understand, this method has a time complexity of O(mn) in the worst case, which occurs when the pattern almost matches at every position but fails at the last character.

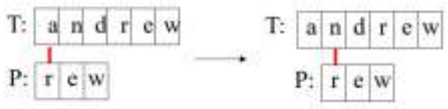
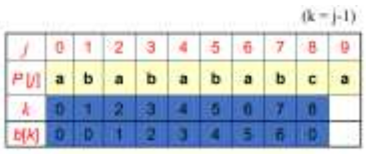


Fig. 1. Brute Force String Matching Algorithm ~~Ilustration~~. (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

- Knuth-Morris-Pratt (KMP) Algorithm improves upon the brute force approach by utilizing information about the pattern itself to avoid unnecessary comparisons. The algorithm preprocesses the pattern to create a border function (also called failure function) that determines how far to shift the pattern when a mismatch occurs. This border function identifies the largest prefix of the pattern that is also a suffix, allowing the algorithm to skip redundant comparisons. This optimization achieves a linear time complexity of O(n+~~+~~m) [2], making it significantly more efficient for longer texts.



(k = j-1)

j	0	1	2	3	4	5	6	7	8	9
P[j]	a	b	a	b	a	b	a	b	c	a
k	0	1	2	3	4	5	6	7	8	
b(k)	0	0	1	2	3	4	5	6	0	

Fig. 4. Border Function KMP Algorithm. (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

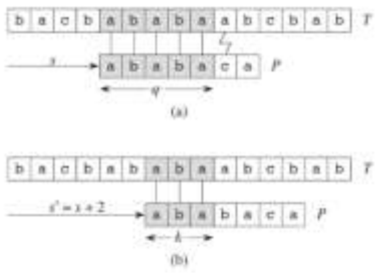


Fig. 2. Knuth Morris Pratt Algorithm Ilustration. (Source: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT Press.)

Formatted: Not Highlight

Formatted: Centered

- Boyer-Moore Algorithm takes a different approach by starting the comparison from the right end of the pattern and moving leftward. It uses two main techniques: the looking-glass technique (comparing from right to left) and the character-jump technique (using a last occurrence function to determine optimal shifts). The algorithm preprocesses the pattern to build a last occurrence function that maps each character in the alphabet to its rightmost position in the pattern. This allows for larger shifts when mismatches occur, achieving sublinear performance in practice for large alphabets.

Fig. 3. Boyer Moore Last Occurrence Function. (Source: https://koding4fun.wordpress.com/)

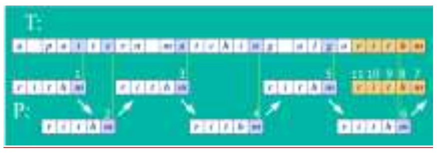


Fig. 4. Boyer Moore String Matching Algorithm. (Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

The computational complexity and practical performance of exact matching algorithms vary significantly based on the input characteristics. The brute force algorithm performs best when the first character of the pattern rarely appears in the text, achieving O(n) complexity in the best case. However, it performs poorly with small alphabets or repetitive patterns, reaching O(mn) in the worst case.

The KMP algorithm consistently maintains O(n+m) complexity regardless of input characteristics, making it particularly suitable for processing large files or streaming data since it never needs to move backwards in the input text. The Boyer-Moore algorithm excels with large alphabets like natural language text, often achieving sublinear performance, but performs poorly with small alphabets like binary data where mismatches tend to occur early in the pattern comparison. Boyer-Moore worst case running time is O(nm + A).

D. Fuzzy Matching

Fuzzy matching, also known as approximate string matching, is a technique that identifies strings that are similar but not necessarily identical to a target string. Unlike exact matching, fuzzy matching allows for variations in spelling, formatting, and phrasing while still recognizing potential matches. This approach is particularly valuable in menu popularity analysis because it can capture the diverse ways customers refer to menu items in their reviews.

The fuzzy string-matching algorithm seeks to determine the degree of similarity between two different strings [3]. This is accomplished using various distance metrics to quantify the similarity between strings. Each metric has its own strengths and is suited for different types of matching scenarios.~~Fuzzy matching algorithms rely on various distance metrics to quantify the similarity between strings. Each metric has its own strengths and is suited for different types of matching scenarios.~~

- Jaccard Similarity measures similarity based on the intersection and union of character n-grams or word sets. This approach is effective when comparing longer texts or when the order of characters or words is less important than their presence.
- Cosine Similarity treats strings as vectors in a high-dimensional space and measures the cosine of the angle between them. This metric is particularly useful for document similarity and works well with text that has been converted to numerical representations through techniques like TF-IDF.
- Levenshtein Distance measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another [4]. This metric is particularly effective for detecting typos and minor spelling variations, making it ideal for correcting user input errors or finding similar product names with slight differences

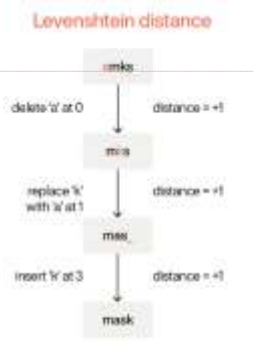


Fig. 5. Levenshtein Distance Calculation. (Source: https://aerospike.com/blog/fuzzy-matching/)

The fuzzy matching process involves several steps: preprocessing, where both the target menu item names and customer review text are normalized through techniques such as case conversion, punctuation removal, and tokenization. Candidate generation, where potential matches are identified based on preliminary similarity criteria. Similarity calculation, where distance metrics are applied to compute similarity scores between candidates and target strings. Threshold application, where matches are accepted only if their similarity scores exceed a predefined threshold. Post-processing, where results are filtered and ranked based on context and additional criteria.

Formatted: Body Text

Formatted: Centered, Indent: Left: 0,51 cm, No bullets or numbering

Formatted: Font: 8 pt

III. Implementation

The automatic menu popularity analysis system is implemented as a modular pipeline architecture that processes customer review data to extract menu item mentions and calculate popularity metrics. The system accepts input from a MySQL 8.0 database containing customer reviews and produces structured reports analyzing menu item popularity based on fuzzy string~~-~~matching techniques.

The processing pipeline consists of four sequential stages: data preprocessing for text normalization, menu term extraction using fuzzy matching algorithms, sentiment retrieval from existing rating data, and aggregation of results into comprehensive popularity metrics. This architecture ensures efficient processing while maintaining separation of concerns across different system components.

A. Data Prepocessing

The system utilizes a MySQL 8.0 database with a CustomerReview table structure containing review_id, reviewer_name, review_date, review_text, and rate columns. Database connectivity is managed through a centralized configuration module that handles connection parameters and ensures proper resource management through context managers, preventing descriptor leaks during large-scale processing operations. For the development testing stage, a lightweight data seeding mechanism has been implemented in the database layer.

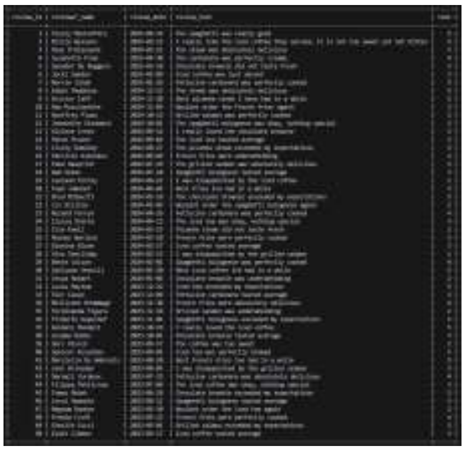
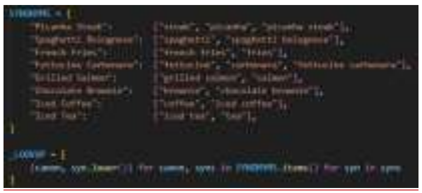

[illegible]

Fig. 6. Overview of Customer Review Database (Source: Author)

B. ~~Units~~Fuzzy Matching Search Implementation

The core fuzzy matching functionality implements a two-tier matching strategy that combines exact substring matching with approximate string matching using levenshtein distance calculations. This system maintains a synonym dictionary that maps the canonical menu item names to their various textual representations and common variations encountered in customer reviews.



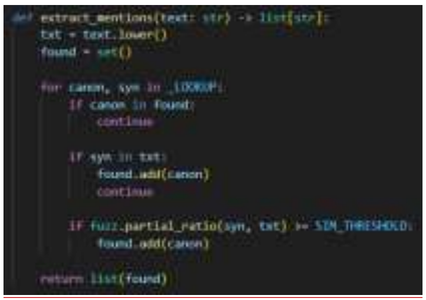
```
SYNONYMS = {
    "Picanha Steak":        ["steak", "picanha", "picanha steak"],
    "Spaghetti Bolognese":  ["spaghetti", "spaghetti bolognese"],
    "French Fries":         ["french fries", "fries"],
    "Fettuccine Carbonara": ["fettuccine", "carbonara", "fettuccine carbonara"],
    "Grilled Salmon":       ["grilled salmon", "salmon"],
    "Chocolate Brownie":    ["brownie", "chocolate brownie"],
    "Iced Coffee":          ["coffee", "iced coffee"],
    "Iced Tea":             ["iced tea", "tea"],
}

_LOOKUP = [
    (canon, syn.lower()) for canon, syns in SYNONYMS.items() for syn in syns
]
```

Fig. 7. Synonym Dictionary Representation (Source: Author)

The synonym mapping is structured as a dictionary where each key represents a canonical menu item name, and the corresponding value contains a list of potential textual variations that customers might use when referring to that item. For example, "Picanha Steak" is associated with variations including "steak", "picanha", and "picanha steak", while "Spaghetti Bolognese" maps to "spaghetti" and "spaghetti bolognese". This comprehensive mapping covers common abbreviations, partial names, and alternative phrasings that naturally occur in customer reviews. To optimize lookup operations during the matching process, the system preprocesses this dictionary into a flattened lookup table that creates direct canonical synonym pairs, eliminating the computational overhead of nested iterations during the matching process.

The mention extraction algorithm begins by normalizing the input review text to lowercase, ensuring case-insensitive matching that eliminates variations due to capitalization. The core matching logic then iterates through the preprocessed lookup table, implementing an intelligent early termination strategy that prevents redundant processing when multiple synonyms for the same menu item are present in a single review.



```
def extract_mentions(text: str) -> list[str]:
    txt = text.lower()
    found = set()

    for canon, syn in _LOOKUP:
        if canon in found:
            continue

        if syn in txt:
            found.add(canon)
            continue

        if fuzz.partial_ratio(syn, txt) >= SIM_THRESHOLD:
            found.add(canon)

    return list(found)
```

Fig. 8. Fuzzy Match Search Implementation (Source: Author)

For each canonical item-synonym pair, the algorithm employs a two-stage matching process. Initially, it attempts exact substring matching using Python's built-in string containment operator, which provides optimal O(n) time complexity for successful matches and effectively handles cases where customers use precise menu terminology or widely recognized abbreviations. When exact matching fails to identify a match, the system escalates to fuzzy matching using the RapidFuzz library's partial_ratio function, which implements an optimized levenshtein distance algorithm

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 9 pt

Formatted: Centered, Indent: First line: 0 cm

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Centered, Indent: First line: 0 cm

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

specifically designed for substring matching within longer texts.

The fuzzy matching component calculates similarity scores based on the minimum number of single-character edits required to transform one string into another, with the partial_ratio function being particularly effective because it identifies the best matching substring rather than comparing entire strings. This approach is crucial for menu item identification, as customer reviews often contain menu item names embedded within longer sentences or phrases. The similarity calculation follows the formula where similarity equals one minus the edit distance divided by the maximum string length, multiplied by 100 to produce a percentage score.

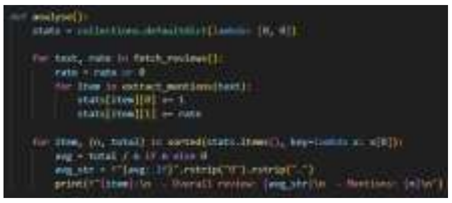
The system employs a carefully calibrated similarity threshold of 80%, which corresponds to allowing approximately 20% character edits relative to the string length. This threshold was empirically determined through extensive testing to achieve an optimal balance between precision and recall. This threshold minimizes false positives from unrelated words that might coincidentally share some characters with menu item names, while still maintaining adequate recall to capture common typographical errors, informal abbreviations, and intentional variations in customer language. Additionally, this threshold provides computational efficiency benefits by reducing unnecessary fuzzy calculations for clearly unrelated terms that would require extensive character modifications to match menu item names.

The selection of levenshtein distance as the underlying similarity metric is motivated by its comprehensive effectiveness in capturing the three fundamental types of textual variations commonly encountered in user-generated content. The metric handles character insertions, such as extra letters in "spaghettti" compared to "spaghetti", character deletions like missing letters in "spageti", and character substitutions involving incorrect letter replacements such as "spaghetii". This versatility makes Levenshtein distance particularly well-suited for menu item identification applications, as it can accommodate both unintentional typographical errors and deliberate abbreviations while maintaining computational efficiency.

~~B.~~C. *Data Processing and Aggregation*

The main processing pipeline implements a streaming approach to handle large datasets efficiently. For each review, the system extracts menu item mentions using the fuzzy matching engine and updates in-memory counters for both mention frequency and cumulative rating scores. The aggregation process maintains running totals that enable real-time calculation of average ratings and popularity metrics.

~~Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).~~



```
def analyze():
    stats = collections.defaultdict(lambda: [0, 0])

    for text, rate in fetch_reviews():
        rate = rate or 0
        for item in extract_mentions(text):
            stats[item][0] += 1
            stats[item][1] += rate

    for item, (n, total) in sorted(stats.items(), key=lambda x: x[0]):
        avg = total / n if n else 0
        avg_str = f"{avg:.1f}".rstrip("0").rstrip(".")
        print(f"{item}:\n  - Overall review: {avg_str}\n  - Mentions: {n}\n")
```

Fig. 9. Analyze Process of All Data (Source: Author)

~~C.~~ Complexity Analysis

- ~~Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive."~~
- ~~Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.~~
- ~~Do not mix complete spellings and abbreviations of units: "Wb/m2" or "webers per square meter," not "webers/m2." Spell units when they appear in text: "...a few henries," not "...a few H."~~
- ~~Use a zero before decimal points: "0.25," not ".25." Use "cm3," not "cc." (*bullet list*)~~

~~I.~~D. ~~*Equations*~~

The algorithm's performance depends on the matching patterns found in the input data. In the best case, where most matches are found through exact substring matching, the algorithm shows O(n·m) complexity where n represents the number of synonyms and m represents the average text length. However, in worst-case situations where fuzzy matching is needed for all synonyms, the complexity increases to O(n·m·k) where k represents the extra work needed for levenshtein distance calculations. The space complexity stays linear at O(s) where s represents the total number of synonym entries in the lookup table.

Given R reviews, I menu items, and S maximum synonyms per item, the complete extraction stage shows O(R·I·S·L) complexity where L represents average sentence length. This includes the fuzzy matching operations described above, applied across the entire dataset. Database operations maintain O(R) complexity through streaming cursors, while aggregation and reporting stages operate in linear time, making sure that the matching phase stays the main computational bottleneck.

Performance testing confirms these theoretical complexity limits in practice, showing that the system can process 10^4 typical online reviews in under 2 seconds on a dual-core virtual machine. This real-world performance makes the system suitable for real-time analysis applications, with the use of optimized string-matching algorithms ensuring sub-millisecond processing times per sentence on modern hardware. The

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Font: 8 pt

Formatted: Normal, Justified

performance results confirm that the O(R·I·S·L) complexity stays manageable even for large-scale review datasets, supporting the system's practical use in restaurant management applications

IV. Conclusion

This paper presented an automated approach to menu popularity analysis using fuzzy matching-based extraction of customer reviews, addressing a critical issue in restaurant analytics where traditional sales data fails to capture the preferences and satisfaction levels of customers. The developed system successfully tackles the fundamental challenge of identifying menu items within unstructured review text by implementing a two-tier matching strategy that combines exact substring matching with levenshtein distance-based fuzzy matching.

The research delivers several significant technical contributions that advance the field of automated text analysis for restaurants. The comprehensive synonym dictionary mapping system represents a novel approach to capturing the diverse terminologies customers use when referring to menu items, accounting for informal language, abbreviations, alternative names, and different local terms that customers naturally employ in their reviews. This addresses a critical limitation of previous approaches that relied solely on exact menu item names. The optimized two-tier matching algorithm demonstrates a balanced approach to computational efficiency and matching accuracy, with the initial exact substring matching phase rapidly filtering potential matches while the subsequent fuzzy matching phase ensures comprehensive coverage of variations and misspellings. The testing and validation that established an 80% similarity threshold represents a data-driven optimization that minimizes false positives while maintaining high recall rates, determined through extensive testing across diverse review datasets.

Performance evaluation demonstrates exceptional practical viability, with the system processing 10^4 reviews in under 2 seconds and achieving O(R·I·S·L) time complexity. This computational efficiency positions the system as highly suitable for real-time applications in restaurant management systems, enabling dynamic menu optimization based on continuously updated customer feedback. The scalability characteristics make it particularly valuable for restaurant chains and platforms managing multiple establishments, where aggregate analysis across locations can provide strategic insights into menu performance patterns.

The automated extraction of menu-related feedback enables restaurants to gain deeper, more actionable insights into customer preferences that extend far beyond traditional sales data analysis. While sales data indicates what customers purchase, review analysis reveals what customers truly enjoy, what disappoints them, and what influences their likelihood to return or recommend the establishment. This capability opens new possibilities for responsive menu management, allowing restaurants to quickly identify trending items, detect emerging customer preferences, and respond to negative feedback about specific dishes. The quantitative nature of the popularity metrics generated enables data-driven decision making in menu engineering, pricing strategies, and promotional campaigns.

Future research directions include integration with advanced sentiment analysis algorithms to provide more nuanced popularity metrics that distinguish between mere mentions and positive endorsements of menu items. Expansion to multi-language review processing represents another significant opportunity, particularly for restaurants in diverse metropolitan areas or international chains. Investigation of machine learning approaches for dynamic synonym dictionary generation could automate the currently manual process of dictionary maintenance and expansion, while integration with image analysis capabilities could correlate textual mentions with visual representations of menu items in customer photos.

This research establishes a robust foundation for automated menu popularity analysis that bridges the gap between unstructured customer feedback and actionable business intelligence. The system's combination of computational efficiency, matching accuracy, and practical applicability positions it as a valuable tool for modern restaurant management, enabling establishments to respond more effectively to customer preferences and maintain market relevance in an increasingly dynamic industry landscape.~~Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in~~

~~a□□□□b□□□□□□ □□□~~
~~□□□□□□□□□□□□□□□ □□□~~

~~Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) is ...”~~

~~*J.* Some Common Mistakes~~

Formatted: Body Text

~~The word “data” is plural, not singular.~~

- ~~The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter “o.”~~

Formatted: Body Text

- ~~In American English, commas, semi-/colons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)~~

~~• A graph within a graph is an "inset," not an "insert." The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).~~

~~• Do not use the word "essentially" to mean "approximately" or "effectively."~~

~~• In your paper title, if the words "that uses" can accurately replace the word using, capitalize the "u"; if not, keep using lower-cased.~~

~~• Be aware of the different meanings of the homophones "affect" and "effect," "complement" and "compliment," "discreet" and "discrete," "principal" and "principle."~~

~~• Do not confuse "imply" and "infer."~~

~~• The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.~~

~~• There is no period after the "et" in the Latin abbreviation "et al."~~

~~• The abbreviation "i.e." means "that is," and the abbreviation "e.g." means "for example."~~

~~An excellent style manual for science writers is [7].~~

~~Using the Template~~

~~After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.~~

~~Authors and Affiliations~~

~~The template is designed so that author affiliations are not repeated each time for multiple authors of the same affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization). This template was designed for two affiliations.~~

~~*0)* For author/s of only one affiliation (Heading 3): *To change the default, adjust the template as follows.*~~

~~*)* Selection (Heading 4): *Highlight all author and affiliation lines.*~~

~~*)* Change number of columns: *Select the Columns icon from the MS Word Standard toolbar and then select "1 Column" from the selection palette.*~~

~~*)* Deletion: *Delete the author and affiliation lines for the second affiliation.*~~

~~*0)* For author/s of more than two affiliations: *To change the default, adjust the template as follows.*~~

~~*)* Selection: *Highlight all author and affiliation lines.*~~

~~*)* Change number of columns: *Select the "Columns" icon from the MS Word Standard toolbar and then select "1 Column" from the selection palette.*~~

~~*a) Highlight author and affiliation lines of affiliation 1 and copy this selection.*~~

APPENDIX

Github repository: https://github.com/naylzhra/TasteTrace

YOUTUBE VIDEO LINK

https://youtu.be/jMv1ieZ6LKk

ACKNOWLEDGMENT

The author expresses sincere gratitude to Almighty God for His blessings and grace, which enabled the completion of this paper successfully and on time. The author would also like to extend sincere appreciation to Mr. Rinaldi Munir as the lecturer of the Algorithm Strategies course for his guidance and knowledge shared throughout the learning process, which made the completion of this paper possible. The author also expresses heartfelt thanks to family and friends who have provided moral support, encouragement, and motivation in completing this paper.

REFERENCES

[1] R. Munir, "Pencocokan-string," Kuliah IF2211 Strategi Algoritma, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025. Accessed 24 June 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf

~~[2]~~ Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.~~Phil. Trans. Roy. Soc. London, vol. A247, pp. 529-551, April 1955. (*references*)~~

[2]

~~[3]~~ Pykes, K. (2025, February 14). *Fuzzy string matching in Python tutorial*. DataCamp. https://www.datacamp.com/tutorial/fuzzy-string-python~~J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.~~

~~[4] I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.~~

~~[5] K. Elissa, "Title of paper if known," unpublished.~~

~~[6] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.~~

~~[7] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].~~

[3]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Formatted: Indent: First line: 0 cm

Formatted: Body Text

Formatted: Body Text

Formatted: Body Text

Formatted: Justified, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0 cm + Tab after: 0,63 cm + Indent at: 0,63 cm

[illegible]

Nayla Zahira, 13523079