



دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارشکار پروژه طراحی الگوریتم فاز 1

زهرا آقایی 40214923

یگانه ظفرزاده 40219303

۱. تعریف مسئله

هدف این پروژه طراحی و تحلیل یک «موتور خلاصه‌سازی ترکیبی» برای متون است که با ترکیب یک روش کلاسیک خلاصه‌سازی الگوریتمی و یک مدل زبانی بزرگ (LLM)، خلاصه‌ای با کیفیت مناسب تولید کند. در این سیستم، بخش الگوریتمی وظیفه استخراج اطلاعات کلیدی را بر عهده دارد و بخش مبتنی بر LLM وظیفه تولید خلاصه‌ای روان‌تر و بازنویسی شده را انجام می‌دهد.

تعریف رسمی مسئله:

یک متن ورودی (T) داده شده است که از (n) جمله تشکیل شده است:

$$T = \{S_1, S_2, \dots, S_n\}$$

هدف، تولید یک خلاصه (S) با طول محدود است؛ به‌طوری‌که خلاصه بتواند مفاهیم مرکزی متن را پوشش دهد، از تکرار اطلاعات جلوگیری کند و تا حد امکان انسجام معنایی و خوانایی مناسبی داشته باشد. طول خلاصه می‌تواند به صورت یکی از حالت‌های زیر تعیین شود:

- انتخاب (k) جمله از متن (خلاصه‌ی جمله‌محور)، یا
- محدودیت درصدی از طول متن (مثلاً ۲۰٪ متن).

خروجی نهایی سیستم از سه جزء تشکیل می‌شود:

۱. **خلاصه الگوریتمی کلاسیک (Extractive):** انتخاب مجموعه‌ای از جمله‌های مهم از متن اصلی (بدون بازنویسی).

۲. **خلاصه مبتنی بر LLM (Abstractive):** تولید خلاصه بازنویسی شده و مفهومی توسط مدل زبانی.

۳. **خلاصه نهایی ادغام شده:** خروجی مرحله‌ای که در آن دو خلاصه فوق با یک سیاست ادغام/مقایسه ترکیب شده و خلاصه نهایی تولید می‌شود (طراحی دقیق ادغام در ادامه پروژه تکمیل خواهد شد).

ورودی سیستم یک متن خام است و خروجی آن یک خلاصه کوتاه‌شده از همان متن می‌باشد که قابلیت ارائه و ارزیابی دارد.

2. تحلیل مسئله

مسئله‌ی خلاصه‌سازی متن از دیدگاه طراحی الگوریتم، قابل مدل‌سازی به صورت یک مسئله روی رشته‌ها و در بخش الگوریتم کلاسیک به صورت یک مسئله گراف محور است. برای پیاده‌سازی بخش کلاسیک، از الگوریتم **TextRank** استفاده می‌شود که در آن جمله‌ها به شکل گره‌های یک گراف در نظر گرفته می‌شوند و یال‌ها میزان شباهت بین جمله‌ها را نمایش می‌دهند.

2.1 دسته‌بندی مسئله از دید الگوریتمی

- پردازش متن / رشته‌ها (**String Processing**): چون ورودی، متن و جمله‌ها هستند و پردازش روی توکن‌ها و جملات انجام می‌شود.
- گراف (**Graph-based**): **TextRank** در یک گراف وزن‌دار بین جمله‌ها ساخته می‌شود. هر جمله یک رأس است و وزن یال‌ها برابر میزان شباهت دو جمله است. سپس یک الگوریتم رتبه‌بندی شبیه **PageRank** روی این گراف اجرا می‌شود تا اهمیت هر جمله تعیین گردد.

2.2 مدل گرافی در **TextRank**

فرض می‌کنیم (n) جمله داریم. یک گراف ($G = (V, E)$) تعریف می‌کنیم که:

- $V = \{s_1, s_2, \dots, s_n\}$
- برای هر دو جمله (s_i) و (s_j) ، اگر شباهت آن‌ها مثبت باشد، یک یال وزن‌دار با وزن (w_{ij}) در نظر گرفته می‌شود.
- معیار شباهت می‌تواند مبتنی بر اشتراک واژه‌ها، TF-IDF و شباهت کسینوسی و روش‌های مشابه باشد (در فاز اول هدف اصلی طراحی و تحلیل است و جزئیات دقیق معیار شباهت می‌تواند در فاز دوم نهایی شود).

ایده اصلی **TextRank** این است که:

<> جمله‌ای مهم‌تر است که به جمله‌های مهم بیشتری شباهت داشته باشد <<

بنابراین، با اجرای یک فرآیند تکرارشونده رتبه‌بندی، به هر جمله یک امتیاز داده می‌شود و جمله‌های با امتیاز بالاتر به عنوان خلاصه انتخاب می‌شوند.

2.3 نقش LLM در مسئله

در این پژوهه، LLM به عنوان یک مؤلفه کمکی (Oracle-like Component) در نظر گرفته می‌شود که صرفاً یک خلاصه‌ی دوم از متن تولید می‌کند LLM. جایگزین الگوریتم کلاسیک نیست و در بخش تصمیم‌گیری الگوریتمی (مثل محاسبه شباخت، رتبه‌بندی و انتخاب جمله‌ها) دخالتی ندارد. هدف از استفاده LLM تولید خلاصه‌ای **Abstractive** است که از نظر خوانایی و روانی بهتر باشد و بتوان آن را با خلاصه‌ی استخراجی TextRank مقایسه و سپس ادغام کرد.

2.4 سختی محاسباتی و ویژگی‌های مسئله

- این مسئله در حالت کلاسیک (TextRank) عموماً NP-Hard محسوب نمی‌شود و یک فرآیند رتبه‌بندی روی گراف است.
- مهم‌ترین چالش الگوریتمی، هزینه محاسبه شباخت برای همه جفت جمله‌ها و اجرای تکرارشونده رتبه‌بندی است.
- خروجی LLM می‌تواند غیرقطعی باشد (ممکن است در اجراهای مختلف کمی تفاوت داشته باشد)، بنابراین وجود بخش الگوریتمی کلاسیک باعث ایجاد یک پایه‌ی پایدار و قابل تحلیل در سیستم می‌شود.

3. فرضیات و محدودیتها

در این پژوهه، فرضیات و محدودیت‌های زیر در نظر گرفته می‌شود:

- متن ورودی به صورت متن ساده (**Plain Text**) ارائه می‌شود.
- متن ورودی دارای جمله‌بندی صحیح بوده و امکان تفکیک آن به جمله‌ها وجود دارد.
- طول خلاصه از پیش مشخص است و می‌تواند به صورت:
 - تعداد ثابتی از جمله‌ها، یا
 - درصدی از طول متن ورودی تعیین شود.
- خلاصه‌سازی الگوریتمی به صورت Extractive انجام می‌شود؛ یعنی خروجی آن شامل انتخاب جمله‌ها از متن اصلی بدون بازنویسی است.
- خلاصه‌سازی مبتنی بر مدل زبانی (LLM) به صورت Abstractive انجام می‌شود؛ یعنی خلاصه‌ی تولیدشده می‌تواند بازنویسی مفهومی متن باشد.

- کیفیت خروجی LLM ممکن است دارای عدم قطعیت یا خطای معنایی باشد و کاملاً قطعی نیست.
- سیستم برای متون بسیار کوتاه (کمتر از چند جمله) بهینه نشده است.

4. طراحی الگوریتم

طبق داک پروژه مسیر *Algorithmic Summarization Engine*, باید حداقل یک الگوریتم کلاسیک داشته باشیم و بهتر است چند روش را مقایسه کنیم. سه روش کلاسیک که می‌توان برای خلاصه‌سازی الگوریتمی استفاده کرد عبارت‌اند از:

- مبتنی بر فراوانی کلمات **Frequency-based Summarization**
 - رتبه‌بندی ساده جمله‌ها **Sentence Ranking**
 - الگوریتم اصلی انتخاب شده **TextRank (Graph-based)**
- در ادامه ابتدا دو الگوریتم فرعی را خیلی خلاصه معرفی می‌کنم و سپس TextRank را مفصل توضیح می‌دهم.

4.1 دو الگوریتم فرعی (برای مقایسه)

Frequency-based Summarization

ایده: کلماتی که در متن زیاد تکرار می‌شوند (غیر از کلمات خیلی عمومی مثل «است»، «و»، ...) معمولاً حامل موضوع اصلی‌اند. جمله‌هایی که تعداد بیشتری از این کلمات مهم را دارند امتیاز بالاتری می‌گیرند و برای خلاصه انتخاب می‌شوند.

- مزیت: ساده و سریع
- ضعف: ارتباط بین جمله‌ها و ساختار کلی متن را مدل نمی‌کند (ممکن است جمله‌های پراکنده انتخاب کند)

Sentence Ranking (Rule-based/Feature-based)

ایده: هر جمله بر اساس چند ویژگی ساده امتیاز می‌گیرد (مثلاً طول جمله، وجود کلمات کلیدی، موقعیت جمله در متن، تعداد کلمات مهم). سپس جمله‌های با امتیاز بیشتر انتخاب می‌شوند.

- مزیت: ساده و قابل کنترل
- ضعف: شدیداً به قواعد دستی وابسته است و «مرکزیت مفهومی» جمله‌ها را مثل روش گرافی نمی‌سنجد

4.2 (الگوریتم اصلی) : TextRank

4.2.1) ایده‌ی اصلی:

یک الگوریتم گراف محور برای خلاصه‌سازی Extractive است. Extractive یعنی خلاصه از خود جمله‌های متن اصلی انتخاب می‌شود (بازنویسی نمی‌شود).

ایده‌ی مرکزی TextRank این جمله است:

یک جمله زمانی مهم است که به جمله‌های مهم دیگر شبیه باشد.

این دقیقاً همان ایده‌ی PageRank در وب است:

- در وب: صفحه مهم است اگر صفحات مهم به آن لینک دهند
- در TextRank جمله مهم است اگر جمله‌های مهم به آن "شباخت/ارتباط" داشته باشند

4.2.2) گراف جمله‌ها یعنی چی؟

برای یک متن با (n) جمله:

- هر جمله (s_i) یک گره (Node) است.
- بین هر دو جمله (s_i) و (s_j) یک یال وزن دار می‌سازیم که وزن آن (w_{ij}) نشان‌دهنده شباهت بین جمله‌های است.

پس گراف ما تقریباً یک گراف «کامل» می‌شود (چون هر جمله با همه جمله‌ها مقایسه می‌شود). اگر بخواهیم هزینه را کمتر کنیم، می‌توانیم فقط یال‌هایی را نگه داریم که شباهتشان از یک آستانه بیشتر باشد (در فاز دوم به عنوان بهینه‌سازی).

4.2.3) شباهت جمله‌ها چطور حساب می‌شود؟

TextRank خودش به ما نمی‌گوید similarity را دقیقاً چطور بسازیم، فقط می‌گوید باید وزن یال‌ها «شباهت» باشد.

در پروژه‌های کلاسیک معمولاً از یکی از این‌ها استفاده می‌شود:

- همپوشانی واژه‌ها (Overlap)
- TF-IDF + Cosine Similarity

4.2.4) چرا TextRank نیاز به اجرای تکرارشونده دارد؟

امتیاز جمله‌ها به هم وابسته‌اند:

- اگر جمله A به جمله B شبیه باشد، امتیاز A به امتیاز B ربط پیدا می‌کند
 - و امتیاز B هم ممکن است به امتیاز A و بقیه ربط داشته باشد
- این وابستگی حلقه‌ای باعث می‌شود نتوانیم امتیازها را «یک‌بار برای همیشه» حساب کنیم.
- پس امتیازها را چند بار به روزرسانی می‌کنیم تا به یک حالت پایدار برسند (همگرایی).

4.2.5) امتیازدهی TextRank دقیقاً چه منطقی دارد؟

در هر مرحله، امتیاز یک جمله (s_i) از دو بخش تشکیل می‌شود:

1. یک مقدار پایه (برای اینکه هیچ جمله‌ای صفر نشود)

2. مجموع سهم جمله‌های دیگر که به آن شبیه‌اند

و سهم هر جمله‌ی دیگر (s_j) به (s_i) این‌طور محاسبه می‌شود:

• اگر (s_j) جمله مهمی باشد \Rightarrow سهمش بیشتر

• اگر شباهت (s_j) به (s_i) زیاد باشد \Rightarrow سهمش بیشتر

• اگر (s_j) به خیلی جمله‌ها شبیه باشد \Rightarrow سهمش بین همه تقسیم می‌شود (یعنی سهم هر کدام کمتر می‌شود)

این دقیقاً منصفانه است:
یک جمله‌ی مهم "اعتبارش" را بین همسایه‌هایش پخش می‌کند، نه اینکه همه را یکسان بالا ببرد.

4.2.6 خروجی TextRank چگونه ساخته می‌شود؟

بعد از اینکه امتیازها همگرا شدند:

- جمله‌ها را بر اساس امتیاز مرتب می‌کنیم
- جمله را انتخاب می‌کنیم Top-k

TextRank (سودوکد 5)

1. Input: Text T, summary size k, damping d, max_iter, eps

2. sentences \leftarrow Split T into sentences

3. n \leftarrow number of sentences

4. Create empty weighted graph G

5. for i = 1 to n do

6. Add node i to G

7. end for

8. for i = 1 to n do

9. for j = 1 to n do

10. if $i \neq j$ then

11. $w \leftarrow \text{Similarity}(\text{sentences}[i], \text{sentences}[j])$

12. if $w > 0$ then

-
13. Add edge $(i \rightarrow j)$ with weight w to G
14. end if
15. end if
16. end for
17. end for
-
18. for $i = 1$ to n do
19. $\text{score}[i] \leftarrow 1 / n$
20. end for
-
21. for $\text{iter} = 1$ to max_iter do
22. $\text{max_change} \leftarrow 0$
23. for $i = 1$ to n do
24. $\text{new_score}[i] \leftarrow 1 - d$
25. for each j such that $(j \rightarrow i)$ exists in G do
26. $\text{new_score}[i] \leftarrow \text{new_score}[i] +$
 $d \times (\text{weight}(j \rightarrow i) / \text{sum_outgoing_weights}(j)) \times \text{score}[j]$
27. end for
28. $\text{max_change} \leftarrow \max(\text{max_change}, |\text{new_score}[i] - \text{score}[i]|)$
29. end for
30. $\text{score} \leftarrow \text{new_score}$
31. if $\text{max_change} < \text{eps}$ then break

32. end for

33. Sort sentences by score in descending order

34. Select top k sentences as summary

35. Output selected sentences

5.1) توضیح سودوکد

خطوط ۱ تا ۳ — ورودی‌ها و جمله‌بندی

- خط ۱: ورودی‌های الگوریتم را می‌گیریم: متن، اندازه خلاصه، پارامترها
- خط ۲: متن را به جمله‌ها تبدیل می‌کنیم
- خط ۳: تعداد جمله‌ها را به دست می‌آوریم

خطوط ۴ تا ۷ — ساخت گراف و اضافه کردن گره‌ها

- خط ۴: یک گراف وزن‌دار خالی می‌سازیم
- خط ۵-۷: برای هر جمله یک گره در گراف اضافه می‌کنیم

خطوط ۸ تا ۱۷ — محاسبه شباهت‌ها و ساخت یال‌ها

- خط ۸-۹: همه جفت‌های جمله‌ها را بررسی می‌کنیم دلیل($O(n^2)$)
- خط ۱۰: جمله با خودش مقایسه نمی‌شود
- خط ۱۱: شباهت دو جمله حساب می‌شود
- خط ۱۲-۱۴: اگر شباهت مثبت بود، یال وزن‌دار اضافه می‌کنیم

خطوط ۱۸ تا ۲۰ — مقداردهی اولیه امتیازها

- ابتدا همه جمله‌ها امتیاز یکسان می‌گیرند (نقطه شروع)

خطوط ۲۱ تا ۳۲ — بخش تکرارشونده (TextRank هسته)

- خط 21: الگوریتم را تا حد اکثر `max_iter` تکرار می کنیم
- خط 22: `max_change`: را صفر می کنیم تا تغییرات را بسنجیم
- خط 23-29: برای هر جمله، امتیاز جدید حساب می شود
- خط 24: مقدار پایه $(1-d)$ را قرار می دهیم
- خط 25: روی جمله هایی می رویم که به جمله i وصل اند (همساخه ها)
- خط 26: سهم هر همسایه j را اضافه می کنیم:

 - اگر j امتیاز بالا دارد \Rightarrow اثر بیشتر
 - اگر وزن شباهت $i \rightarrow j$ بالا است \Rightarrow اثر بیشتر
 - اگر j به خیلی ها وصل است \Rightarrow سهم تقسیم می شود

- خط 28: تغییر امتیاز را ثبت می کنیم
- خط 30: بعد از محاسبه همه، امتیازها را آپدیت می کنیم
- خط 31: اگر تغییرات خیلی کم شد (همگرایی) حلقه را می شکنیم

خطوط 33 تا 35 – انتخاب خلاصه

- جمله ها را بر اساس امتیاز مرتب می کنیم و `top-k` را خروجی می دهیم

6. تحلیل زمان و فضا

در این بخش، پیچیدگی زمانی و فضایی الگوریتم خلاصه سازی الگوریتمی مبتنی بر **TextRank** تحلیل می شود.

فرض می کنیم:

- n = تعداد جمله های متن ورودی
- $(iter)$ = تعداد تکرارهای الگوریتم **TextRank** تا همگرایی

1. پیش‌پردازش و جمله‌بندی متن**عملیات:**

- تقسیم متن به جمله‌ها
- حذف علائم نگارشی و توقف‌واژه‌ها (در صورت استفاده)

پیچیدگی زمانی: $O(n)$ **پیچیدگی فضایی:** $O(n)$ **2. ساخت گراف شباهت جمله‌ها**

در این مرحله، برای هر جفت جمله، میزان شباهت محاسبه می‌شود.

عملیات:

- مقایسه همه جفت‌های جمله‌ها
- محاسبه وزن یال‌ها

پیچیدگی زمانی: $O(n^2)$ **پیچیدگی فضایی:** $O(n^2)$

زیرا گراف به صورت کامل یا نیمه کامل ذخیره می‌شود.

3. مقداردهی اولیه امتیاز جمله‌ها

به هر جمله مقدار اولیه مساوی داده می‌شود.

پیچیدگی زمانی:

$O(n)$

پیچیدگی فضایی:

$O(n)$

4. اجرای الگوریتم (Iterative) TextRank

در هر تکرار:

- امتیاز هر جمله با توجه به جمله های مرتبط به آن محاسبه می شود
- این محاسبه برای همه جمله ها انجام می شود

پیچیدگی زمانی هر تکرار:

$O(n^2)$

تعداد تکرارها:

- معمولًاً ثابت و کوچک (مثلاً 20 تا 30)

پیچیدگی زمانی کل:

$O(\text{iter} * n^2)$

پیچیدگی فضایی:

$O(n)$

فقط بردار امتیازها نگهداری می شود.

5. مرتب‌سازی نهایی جمله‌ها

پس از محاسبه امتیاز نهایی، جمله‌ها بر اساس امتیاز مرتب می‌شوند.

پیچیدگی زمانی:

$$O(n \log n)$$

پیچیدگی فضایی:

$$O(n)$$

پیچیدگی کلی الگوریتم

پیچیدگی زمانی نهایی:

$$O(\text{iter} * n^2)$$

زیرا جمله‌ی غالب مربوط به ساخت گراف و اجرای TextRank است.

پیچیدگی فضایی نهایی:

$$O(n^2)$$

به دلیل ذخیره گراف شباهت جمله‌ها.

تحلیل بهترین، بدترین و میانگین حالت

- بهترین حالت:

متن کوتاه با تعداد جمله کم => اجرای سریع

- بدترین حالت:

متن طولانی با تعداد جمله زیاد => هزینه‌ی (n^2) بالا

- حالت میانگین:

متوسط با اندازه متوسط => قابل اجرا در زمان مناسب

7. نقش مدل زبانی بزرگ (LLM) در سیستم

در این پژوهه، مدل زبانی بزرگ (LLM) به عنوان یک مولفه‌ی کمکی در کنار الگوریتم کلاسیک خلاصه‌سازی استفاده می‌شود و جایگزین الگوریتم اصلی نیست. هدف از بهکارگیری LLM افزایش کیفیت زبانی و روانی خلاصه نهایی است، در حالی که ساختار تصمیم‌گیری الگوریتمی سیستم حفظ می‌شود.

در معماری سیستم، خلاصه‌سازی در دو مرحله مستقل انجام می‌گیرد:

1. خلاصه‌سازی الگوریتمی کلاسیک (TextRank)

این مرحله به صورت extractive عمل کرده و جمله‌های مهم متن را بر اساس تحلیل گرافی و رتبه‌بندی الگوریتمی انتخاب می‌کند.

2. خلاصه‌سازی مبنی بر LLM

در این مرحله، LLM با دریافت متن ورودی، یک خلاصه abstractive و بازنویسی شده تولید می‌کند که از نظر زبانی روان‌تر و منسجم‌تر است.

مدل زبانی در این پژوهه نقش تصمیم‌گیرنده ندارد و در هیچ‌یک از مراحل زیر دخالت نمی‌کند:

- محاسبه شباهت بین جمله‌ها
- رتبه‌بندی یا امتیازدهی جمله‌ها
- انتخاب جمله‌ها برای خلاصه الگوریتمی

به عبارت دیگر، LLM به عنوان یک Oracle در نظر گرفته می‌شود که تنها یک خروجی زبانی تولید می‌کند و فرآیند داخلی آن در تحلیل الگوریتمی لحاظ نمی‌شود. خروجی LLM در فاز بعدی پژوهه، توسط یک الگوریتم ادغام با خلاصه الگوریتمی ترکیب خواهد شد تا خلاصه نهایی سیستم تولید شود.

به دلیل ماهیت غیرقطعی LLM، خروجی آن ممکن است در اجراء‌های مختلف کمی مقاوت باشد یا برخی جزئیات را حذف کند؛ از این‌رو، وجود بخش الگوریتمی کلاسیک به عنوان پایه‌ای پایدار و قابل تحلیل برای سیستم ضروری است.

