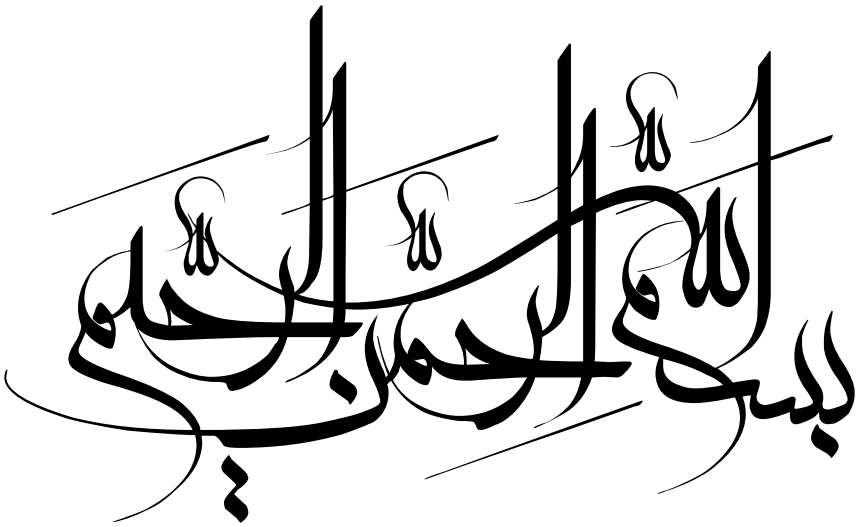

دانیال رومیانی – سینا دالوند

مقدمه :

اهداف کلی این پژوهش به شرح زیر هستند :

۱. در قدم اول باید ۱۴۰۰ داکیومنت مورد نیاز را از مجموعه کالکشن ها جدا کنیم با توجه به این که فایل کالکشن ما از نوع متنی بوده اولین کار نوشتن یه جدا کننده متنی (parser) است که تعداد ۱۴۰۰ داکیومنت را برای ما از کالکشن بیرون بکشد.

۲. گام دوم ساخت ایندکس برای هر داکیومنت است (Indexing) که به چهار روش زیرصورت مگیرد :

Without stemming <=

With stemming<=

Without stop words <=

Whit stop words<=

و در نهایت روش های امتیاز دهی اسناد هم با اعمال کردن روش های امتیاز دهی روی نتایج ارزیابی برای چهار ایندکس صورت میگیرد.

۳.معیار های ارزیابی خواسته شده برای این گزارش (NDCG , MAP , P@10 , P@5 هستند که محاسبه معیار های خواسته شده توسط ابزار TrecEval انجام میشود .

بدین صورت که خروجی الگوریتم بالا به عنوان ورودی TrecEval قرار داده میشود و خروجی TrecEvla هم معیار های ارزیابی هستند که در ادامه به صورت کامل توضیح داده میشوند.

تمامی نتایج محاسبات در پوشه src ➔main➔ resources موجود میباشد (هم فایل ایندکس هم منابع اصلی و هم نتیجه سرچ ها و نتیجه ارزیابی هر روش)

که نام فایل ها ترکیبی از نام تحلیلگر(analyzer) و روش امتیاز دهی(similarity) میباشد .

در هنگام اجرا هم زمان صرف شده توسط هر عملیا تبه واحد میلی ثانیه اعلام میگردد .

1 - Parser :

در این قسمت هدف جداسازی ۱۴۰۰ داکیومنت از فایل cran.all.1400 که همان فایل کالکشن هاست میباشد .

فایل کالکشن ما به صورت متنی می باشد که داکیونت ها را با استفاده از عبارت ".I" (دات ای) از یک دیگر جدا هستند

به عنوان مثال داکیومن شماره ۲۵ به صورت زیر است :

.I 25

.T

Title

.A

Author

.B

Bibliography

.W

Content

.I 26

که در این ساختار عبارت بعد از ".T" عنوان داکیومت و بعد از ".A" نویسنده و بعد از ".B" چکیده ای از داکیومنت و بعد از ".W" هم متن اصلی داکیومنت است.

برای پارس کردن متن کافیست که با دیدن عبارت ".I" دشروع به اضافه کردن متن به یک رشته (string) کنیم و در نهایت با رویت مجدد ".I" ابتدار شماره (ID) و متن داکیومنت را به ایندکس اضافه کرده و اعمال بالا مجدد تکرار می شوند تا زمانی که پایان کالکشن برسیم و داکیومنتی برای استخراج موجود نباشد.

که قطعه کد آن به صورت زیر است (با زبان کاتلین) :

```
fun indexDocument(filePath: String, writer:
IndexWriter) {
```

```
    Files.newInputStream(Path.of(filePath)).use {
stream ->
        val buffer =
BufferedReader(InputStreamReader(stream,
StandardCharsets.UTF_8))
        var id = ""
        var title = ""
        var author = ""
        var bib = ""
        var w = ""
        var state: String? = ""
        var first = true
        var line: String? = buffer.readLine()
        while (line != null) {
            when (line.substring(0, 2)) {
                ".I" -> {
                    if (!first) {
                        val d: Document =
makeDocument(id, title, author, bib, w)
                        writer.addDocument(d)
                    } else {
                        first = false
                    }
                    title = ""
                    author = ""
                    bib = ""
                    w = ""
                    id = line.substring(3,
line.length)
                }
                ".T", ".A", ".B", ".W" -> state =
line
                else -> when (state) {
                    ".T" -> title += "$line "
                    ".A" -> author += "$line "
                    ".B" -> bib += "$line "
                    ".W" -> w += "$line "
                }
            }
            line = buffer.readLine()
        }
        val d: Document = makeDocument(id, title,
author, bib, w)
        writer.addDocument(d)
```

```
        }
}
```

که درون بدنه این متد (indexDocument) برای اضافه کردن به ایندکس متد makeDocument فراخوانی میشود.

برای پارس کردن کوئري از پارسر کتابخانه لوسین استفاده کرده واز کلاس MultiFieldQueryParser در این جهت اسفتاده شده است که ساختار کوئري هم همانند داکیومتن ایندکس شود (برای مقایسه راحت تر) سپس برای آن کوئري علمیات سرچ روی داکیومنت ها انجام شده ونتایج مناسب در خروجی چاپ میشوند و تمام عملیات بالا دوباره اجرا میشود تا زمانی که دیگر کوئري در فایل موجود نباشد.

۲ – نمایه سازی (Indexing) :

برای نمایه سازی از کتابخانه متن باز لوسین استفاده میکنیم . با کمک از چهار مدل تحلیل گر متن نمایه ساز های این کتابخانه نمایه های متناظر با هر داکیومنت را میسازند.

۱. بدون Stop Words (StandardAnalyzer) : که کلمات ساده و بدون اثر و کم اهمیت در موضوع مورد جست و جو و در نتیجه آن مانند ...,a , in , an , the را در نظر نمیگیرد .
۲. همراه با Stop Words (simpleAnalyzer) : که در این تحلیل گر هیچ کلمه و حرفی را حذف نمیکند.

۳. بدون Stemming (whitespaceAnalyzer) : که هیچ گونه عملیات سبک سازی از جهت ریشه کلمات را اعمال نمیکند.

۴. همراه با Stemming (EnglishAnalyzer) : که برای کلماتی که در یک خانواده هستند یک کلمه متناظر در نظر میگیرد .

که در منویی در ابتدای اجرای برنامه امکان انتخاب تحلیلگر فراهم شده است (در متد analyzerMenu)

از جهت حجم فایل ایندکس :

نام روش	حجم فایل ایندکس
StopWords	1.38MB
non-StopWords	1.45MB
Stemming	1.57MB
non-Stemming	1.32MB

کمترین سایز ایندکس در روش بدون stemming بوده که بعلت ریشه یابی کلمات اسناد این مقدار قابل پیش بینی بود. در مقایسه میزان حافظه مصرفی مشاهده میشود که کمترین سایز متعلق به روش بدون stemming است که مانند معیار قبلی مقایسه قابل قبولی است. در آخرین مقایسه مشاهده میشود که سریع ترین نمایه در روش بدون Stemming و word Stopساخته میشود که به نظر میرسد الگوریتم اجراي سريع تري نسبت به روش هاي قبلی باید داشته باشد چرا که درگیر جداسازي کلمات اضافه و ریشه یابی کلمات نخواهدشد

۲ – روش های امتیاز دهی :

۱. پیش فرض لوسین : امتیاز دهی پیش فرض لوسین است که امتیاز هر سند را بصورت پیش فرض براي هر کوئري تعیین میکند و رتبه بندي اسناد را با توجه به امتیازي که به اسناد میدهد انجام میدهد.
۲. Smoothing (JM) : gروش هاي هموارسازي سعی می کنند احتمال رخدادهاي دیده نشده را به نحوي تخمین بزنند.که ما در این جا از دو مدل JM و Drichlet استفاده کردیم .در متد JM از ترکیب خطی اسناد مدل هاي زبانی با پس زمینه زبان رایج استفاده می کند که رابطه آن فرمول زیر است

$$P[q|\theta] = \lambda P[q|d] + (1-\lambda)P[q|C]$$

که پارامتر λ را برابر ۵ .در نظر گرفتیم. و از کلاس LMJelinekMercerSimilarity یرای پیاده سازی استفاده کرده ایم.

۳. Smoothing (Drishlet) : در متد Dirichlet حداقل وزن را به مجموعه داده ها می دهد و وزن بیشتر را به اسناد می دهد و اسناد طولانی تر Max likelihood بهتري نسبت به اسناد کوتاهتر داشتند .
رابطه این متد فرمول زیر است:

$$\hat{p}_j(d) = \lambda P[j|d] + (1-\lambda) P[j|C]$$

۴. TF-Idf پیش فرض : که به صورت زیر پیاده سازی میشوند:
کلاسی که از TFIDFSimilarity ارث بری کرده و متد های زیر را لغو میکند:

```
5.
  override fun tf(freq: Float): Float = ln((
  freq).toDouble()).toFloat()+1.toFloat()

  override fun idf(docFreq: Long, docCount: Long):
  Float
  {
      var a=ln((docFreq / (docCount +
  1)).toDouble()).toFloat()
      if (a>0) return a else return 0.toFloat()
  }

  override fun lengthNorm(length: Int): Float = 1f

  override fun sloppyFreq(distance: Int): Float = 1f

  override fun scorePayload(doc: Int, start: Int,
  end: Int, payload: BytesRef?): Float = 1f
```

روش ارزیابی همانند آنالیزور ها در ابتدای اجرای برنامه توسط منویی قابل انتخاب هستند متد (similarityMenu) و بعد از انتخاب شدن توسط قطعه کد های زیر هم در کلاس ایندکس کننده و جست وجو کننده در IndexWriterConfig و IndexSearcher ست میشوند.

متد (getSimilarity) در کلاس های ایندکس(Indexer) و سرچ (Indexer) بدین گونه فرخوانی میشود :

Indexer ➔ index :

```
val iwc =
IndexWriterConfig(getAnalyzer(analyzer)).apply {
    openMode = OpenMode.CREATE
    getSimilarity(similarity)?.let {
        this.similarity = it
    }
```

Searcher ➔ Search :

```
val searcher = IndexSearcher(indexes).apply {
getSimilarity(similarity)?.let {
this.setSimilarity(it) } }
```

۳ – روش های ارزیابی :

ارزیابی روشهاي مختلف با معیارهاي اندازه گیري MAP , P@10 , P@5 , NDCG در اهداف این پروژهش هستندکه در ادامه به نحوه محاسبه آنها میپردازیم.

برای پیاده سازی ارزیابی از Trec Eval استفاده شده که پروژه ای به زبان c میباشد و تمامی میعار های ارزیابی مورد نیاز ما را محاسبه خواهد کرد.

بدین صورت که ما فایل آیده آل خود را (که در پژوهش ما (cranqrel) نام دارد) به همراه خروجی مرحله قبل (که شامل فایل خروجی نتیجه سرچ است) را به این برنامه داده و در نهایت با ایجاد فایلی متنی خروجی محاسبات را درآن ذخیره میکنیم (نام فایل همان نام خروجی + eval- می باشد)

برای استفاده از آن پکیج trec eval(uk.ac.gla.terrier.jtreceval.trec_eval) را به برنامه اضافه کرده و در تابع سازنده کلاس خود با استفاده از قطعه کد زیر دستور محسابه مقادیر (-m map -m ndcg -m p.5,10) را وارد میکنیم :

```
trecEval(results: String) = timmy("Trec Eval Done
in") {
    val te = trec_eval()
    val output = te.runAndGetOutput(arrayOf("-m",
"map", "-m", "ndcg", "-m", "P.5,10",
Const.newqrelPath, results))
    saveEvalResult(output,
generateEvalOutputName(results))
}
```

نکته قایل توجه در مورد استفاده این پکیج این است که با تشخیص نوع سیستم عامل ، فایل کامپایل شده و مناسب Trec evalمخصوص سیستم عامل شما را مورد استفاده قرار میدهد.

در جدول زیر هم تمامی معیار های ارزیابی روش های مختلف را مشاهده میکنید :

		Default lucene	Smoothing(JM)	Smoothing (Drishlet)
Stemming ■ Stop word ■	NDCG	0.0259	0.0247	0.0233
	P@10	0.0043	0.0032	0.0032
	P@5	0.0043	0.0043	0
	MAP	0.0043	0.0035	0.0033
Stemming ■ Stop word □	NDCG	0.0255	0.0245	0.0229
	P@10	0.0043	0.0032	0.0032
	P@5	0.0043	0.0043	0
	MAP	0.0043	0.0035	0.0033
Stemming □ Stop word ■	NDCG	0.0239	0.268	0.0240
	P@10	0.0053	0.0053	0.0053
	P@5	0.0064	0.0043	0.0043
	MAP	0.0037	0.0036	0.0034
Stemming □ Stop word □	NDCG	0.0236	0.0265	0.0237
	P@10	0.0053	0.0053	0.0053
	P@5	0.0064	0.0043	0.0042
	MAP	0.0037	0.0036	0.0033