МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической и прикладной информатики

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине: Программные системы статистического анализа

на тему: Исследование способов извлечения ключевых фраз из текста с помощью пакета RKEA

Вариант №57

Факультет: ФПМИ

Группа: ПММ-21

Выполнил: Сухих А.С.

Проверила: к.э.н. Тимофеева А.Ю.

Дата выполнения: 13.01.23

Отметка о защите:

Новосибирск 2023

1. Реализация алгоритма на языке R

RKEA (R implementation of Keywords Extraction Algorithm) – программный пакет для языка программирования R, являющийся интерфейсом для Java-приложением KEA. Применяется для извлечения ключевых слов из текстов. Использует модель Наивного Байеса (обучение с учителем) для определения ключевых слов.

В пакете есть всего две функции:

- createModel(corpus, keywords, model, voc, vocformat) – создание и обучение модели извлечения ключевых слов. corpus – текст некоторого документа (Corpus из пакета tm), keywords – список ключевых слов, присутствующих в документе, model – название модели (будет создан файл конфигурации с данным названием), voc – автоматизационный словарь, vocformat – формат словаря. Последние два параметра не являются обязательными и позволяют производить поиск ключевых слов только из определенного словаря терминов

- extractKeywords(corpus, model, voc, vocformat) – извлечение ключевых слов. corpus – текст, который будет анализироваться, model – название модели, переданной в createModel.

Исходная программа KEA работает только с английским, испанским, немецким и французским языками. В данной работе будет анализироваться английский текст.

Ниже приведен исходный код программы по обучение модели и извлечению ключевых слов из статей новостного издательства Reuters:

```
# подключение библиотек RKEA для извлечения ключевых слов
# и tm для обработки текста
library(RKEA)
library(tm)
library(stringdist)
```

```
# функция чтения ключевых слов из файла
get_keywords <- function(path){
  kw = vector(mode = "list", length = length(list.files(path, pattern = "*.txt")))
  # print(path)
  i = 1
  for (file in list.files(path, pattern = "*.key", full.names = TRUE)){
    # print(file)
    kw[[i]] = c(scan(file, character(), quote = "", sep="\n"))
    i = i + 1
  }
  names(kw) = sapply(strsplit(list.files(path, pattern = "*.key"), '[.]'), '[', 1)
  # print(str(kw))
  return (kw)
}

# функция создания модели и обучения на тренировочных данных
train_model <- function(model_name, data_folder, keywords) {
  path = paste(getwd(), data_folder, sep = '/')
  # print(path)
  # формирование корпуса из текстовых файлов
  corp = Corpus(DirSource(directory = path, pattern = '*.txt'))
  # создание временного файла, хранящего конфигурацию модели
  tmpdir = tempfile()
  dir.create(tmpdir)
  model = file.path(tmpdir, model_name)
  res = createModel(corp, keywords = keywords, model = model, )
  print(res)
  return (model)
}

# функция извлечения ключевых слов из текстовых файлов
extract_keywords <- function(model, data_folder) {
  path = paste(getwd(), data_folder, sep = '/')
  print(path)
```

```
  test_corp = Corpus(DirSource(directory = path, pattern = '*.txt'))
  kw = extractKeywords(test_corp, model)
  names(kw) = sapply(strsplit(names(test_corp), '[.]'), '[', 1)
  return (kw)
}

# функция записи ключевыхслов в файл
write_keywords = function(path, keywords) {
  output_path = sprintf("%s/extracted_kw/", path)
  if (!dir.exists(output_path))
    dir.create(output_path)
  for (i in 1:length(keywords)){
    # print(i)
    write(keywords[[i]],
          paste(output_path, names(keywords)[[i]], '_1.key', sep = ''))
  }
}

keywords_similarity = function(keywords1, keywords2){
  if (length(keywords1) != length(keywords2))
    stop("Lengths of vectors must be equal")

  lv_sim = vector(mode = "double", length = length(keywords1))
  # lv = vector()
  for (i in 1:length(keywords1)){
    # sapply(keywords1, )
    lv = vector(mode = "double", length = length(keywords1[[i]]))
    for (j in 1:length(keywords1[[i]]))
      # нормализуем расстояние Левенштейна
      lv[j] = min(stringdist(tolower(keywords1[[i]][j]), tolower(keywords2[[i]]), method =
"lv") /
                  pmax(nchar(keywords1[[i]][j]), nchar(keywords2[[i]])))
    # получаем сходство строк как обратную величину нормализованному
расстоянию
```

```
    lv_sim[i] = 1 - mean(lv)
  }
  # print(lv_dist)
  # возвращаем усредненное значение сходства по всем документам
  return (lv_sim)
}

perform_test = function(model_name, train_folder, data_folder, true_keywords_folder){
  # train_keywords = get_keywords(paste(train_folder, "kw", sep = '/'))
  train_keywords = get_keywords(train_folder)
  # print(train_keywords)
  training_mem = mem_change({
    training_time = system.time({
      model = train_model(model_name = model_name, data_folder = train_folder,
keywords = train_keywords)
    })
  })

  extract_mem = mem_change({
    extract_time = system.time({
      extracted_kw = extract_keywords(model, data_folder)
    })
  })
  print(extracted_kw)
  write_keywords(data_folder, extracted_kw)
  true_keywords = get_keywords(path = true_keywords_folder)
  print(true_keywords)
  similarity = keywords_similarity(extracted_kw, true_keywords)
  print("Keywords:")
  print(length(extracted_kw))
  print("Training time:")
  print(training_time[3])
  print("Extracting time:")
  print(extract_time[3])
```

```
  print("Training memory:")
  print(training_mem)
  print("Extracting memory:")
  print(extract_mem)
  print("Average similarity:")
  print(mean(similarity))
  unlink(dirname(model), recursive=TRUE)
  return (list(extracted_kw = extracted_kw, true_kw = true_keywords, similarity = similarity))
}

# сгенерированные модельные примеры
model_name = "generated"
train_folder = "generated/repeats_sample/training"
data_folder = "generated/repeats_sample/"
res = perform_test(model_name, train_folder, data_folder, paste(data_folder, "true_kw", sep=''))

# Reuters TG
model_name = "reuters_tg"
train_folder = "reuters_tg/full/test_ds/training"
data_folder = "reuters_tg/full/test_ds/"
res = perform_test(model_name, train_folder, data_folder, data_folder)

# Krapivin Dataset
model_name = "krapivin"
train_folder = "krapivin/training"
data_folder = "krapivin/tmp"
res = perform_test(model_name, train_folder, data_folder, data_folder)

# Customer Churn
model_name = "churn"
train_folder = "krapivin"
data_folder = "articles/tmp"
```

```
res = perform_test(model_name, train_folder, data_folder, data_folder)
```

2. Модельный пример

В качестве модельного примера используется случайно сгенерированный текст. При генерации случайным образом выбирались слова из английского словаря объемом 10000 слов, предоставленного MIT [5]. Также для имитации наличия ключевых слов из текста случайным образом выбирались слова, которые должны повторно появиться в тексте несколько раз в произвольном месте.

Таким алгоритмом генерируется некоторый набор тренировочных текстов и тестовых текстов. Для этих текстов также записываются файлы с ключевыми словами, которые были выбраны случайно в процессе генерации.

3. Реальный пример

Для проверки на реальном примере были взяты краткие выжимки из статьей новостного агентства Reuters с июля 2022 года по 14 января 2023 года. Статьи были выгружены из неофициального телеграм-канала Reuters: World в формате JSON и преобразованы в текстовые файлы, каждый соответствующий своей статье. За данный период была опубликована 321 статья. Некоторые из них были выбраны для обучения модели.

4. Исследование на модельных и реальных примерах.

Исследование на проводилось с различными условиями экспериментов. Для модельных примеров входными условиями были: количество текстов (N), объем текста (n), количество ключевых слов в исходном тексте.

В качестве выходных показателей оценивалось время вычисления, время загрузки файла, занятый объем памяти, сходство полученных ключевых слов с истинными ключевыми словами, оцененное с помощью расстояния Левенштейна.

RKEA не позволяет задавать количество искомых ключевых слов при анализе текста и практически во всех случаях находит 10 ключевых слов.

В модельных примерах используется случайно сгенерированный текст с некоторым числом повторяющихся слов. Пример одного из таких текстов:

necessity motorola belfast sensitive recipient motorola full lip recipient words couple jaguar currency zu words law immune section bingo highs unique bills puppy vip recipient encourage licensing cabinets likely tsunami implement athletes gaps comply recipient removal unions hebrew earl producing egyptian gained viewers motorola highs mortality eve environment freeze kissing motorola highs um cement install utilization regulatory narrow cadillac broader diameter leeds recipient author hay colon demonstration marketing unsubscribe tactics words tactics noon inbox vertical highs expressions divorce dee freeze tits gaps communications calendar timothy describes inbox aspnet oops motorola nu ronald link andorra philippines inbox words tactics inbox words

При генерации в файле ключевых слов были указаны слова *tactics, freeze, jaguar, inbox, hay, words, motorola, highs, recipient, gaps.*

В результате работы программы были извлечены словосочетания *motorola, recipient, words, highs, inbox, tactics, gaps, freeze, tactics words, inbox words tactics.*

Сходство полученных ключевых слов с истинными составило 0.89069.

На данных моделируемых примерах генерируется множество текстов и сходство ключевых слов определяется как среднее между сходствами всех текстов. Количество генерируемых тестовых текстов во всех экспериментах равно 100.

Таблица 1. Исследование извлечения ключевых слов на модельном примере

	Входные данные			Выходные данные				
№	N_{tr}, кол-во тренировочных текстов	n, объем текста	K_{tr}, смоделированных ключевых слов	t_o, время обучения, с	$T_и$, время извлечения, с	mem_o, память при обучении	$mem_и$, память при извлечении	lev, сходство эксп. и теор. ключ. слов
1	10	100	5	0.085	0.328	-41.7 kB	102 kB	0.79227

2	10	100	10	0.118	0.382	14.1 kB	102 kB	0.92903
3	10	100	20	0.087	0.344	6.38 kB	84.8 kB	1
4	50	100	5	0.411	0.333	-40.4 kB	102 kB	0.79286
5	100	100	5	0.85	0.343	-40.8 kB	103 kB	0.77919
6	500	100	5	4.508	0.462	146 kB	91.2 kB	0.82774
7	10	500	10	0.431	1.444	-43.5 kB	85 kB	0.90194
8	10	1000	10	0.868	2.819	-42.5 kB	86.6 kB	0.85223

По результатам видно, что данный способ моделирования не очень показателен, так как достигается абсолютное сходство по расстоянию Левенштейна при превышении количеством смоделированных ключевых слов количества извлеченных. Также в остальных экспериментах значения сходства могут существенно различаться в зависимости от случайно сгенерированных текстах при одних и тех же параметрах.

Рассмотрим реальный текст из статей Reuters. Для данных статей мной вручную были подготовлены ключевые слова для тренировочного датасета и для тестовых текстов. Обучение модели было произведено на 10-50 статьях, тестирование модели выполняли на примере тестовых статей. Также сравним модель, обученную по текстам статей и модель, обученную по модельным примерам

Примеры извлечения текста после обучения:

Текст 1. Ukraine interior minister among dead in helicopter crash that ignites nursery.

Ukraine's interior minister and a child were among at least 14 people killed on Wednesday when a helicopter crashed into a nursery and set it ablaze in a suburb of the capital Kyiv.

Dozens of other people were hurt, including a number of children, many suffering burns. The French-made Super Puma helicopter went down in the fog in Brovary on the eastern outskirts of Kyiv, plummeting into the nursery grounds.

Ukrainian state emergency services said 14 people in total had been killed. Government agencies had earlier published higher death tolls ranging up to 18.

Теоретические ключевые слова: *Ukraine interior minister, dead, people killed, helicopter crashed, helicopter went down, outskirts of Kyiv.*

Ключевые слова, полученные моделью, обученной на статьях (10 текстов): *interior, interior minister, helicopter, nursery, minister, Kyiv, people, killed*.

Ключевые слова, полученные моделью, обученной на модельных примерах: *killed, Kyiv, minister, interior, helicopter, nursery, people, interior minister.*

Текст 2. Sweden, Finland must send up to 130 "terrorists" to Turkey for NATO bid, Erdogan says.

Sweden and Finland must deport or extradite up to 130 "terrorists" to Turkey before the Turkish parliament will approve their bids to join NATO, President Tayyip Erdogan said.

The two Nordic states applied last year to join NATO following Russia's invasion of Ukraine but their bids must be approved by all 30 NATO member states. Turkey and Hungary have yet to endorse the applications.

Turkey has said Sweden in particular must first take a clearer stance against what it sees as terrorists, mainly Kurdish militants and a group it blames for a 2016 coup attempt.

Теоретические ключевые слова: *Sweden, Finland, send up, terrorists, NATO, invasion of Ukraine, Turkey, Erdogan.*

Ключевые слова, полученные моделью, обученной на статьях (10 текстов): *Sweden, terrorists, bid, NATO, Finland, Turkey, Erdogan, invasion, join, join NATO.*

Ключевые слова, полученные моделью, обученной на модельных примерах: *NATO, Turkey, terrorists, Sweden, bid, Finland, Erdogan, invasion, Russia's, said.*

Таблица 2. Исследование извлечения ключевых слов из статей Reuters

Входные данные				Выходные данные				
№ текста	n, объем текста	N_{tr}, кол-во тренировочных текстов	Обучение модели	t_o, время обучения, с	$T_и$, время извлечения, с	mem_o, память при обучении	$mem_и$, память при извлечении	lev, сходство эксп. и теор. ключ. слов
1	646	10	статьи	0.04	0.007	-8.49 kB	3.17 kB	0.40865
		30	статьи	0.116	0.007	6.79 kB	3.17 kB	0.40865
		50	статьи	0.148	0.009	22 kB	3.17 kB	0.40865
		10	модель	0.091	0.007	-72 B	3.17 kB	0.40865
		50	модель	0.425	0.015	22.4 kB	3.17 kB	0.40865
2	653	10	статьи	0.041	0.006	5.96 kB	3.37 kB	0.74369
		30	статьи	0.12	0.008	14.3 kB	3.23 kB	0.74369
		50	статьи	0.243	0.015	22.6 kB	3.91 kB	0.74369
		10	модель	0.093	0.007	-2.36 kB	3.3 kB	0.72925
		50	модель	0.443	0.016	-42.9 kB	3.3 kB	0.67845

Как видно по приведенной выше таблице в RKEA практически никакого влияния на результат не оказывает обучение модели. Даже если обучать модель всего на одном тексте, в котором содержатся несвязанные друг с другом слова, как в модельных примерах, модель всё равно извлекает те же самые слова.

Единственное, было замечено, что при обучении модели на ключевых словах, которые не содержатся в тренировочном тексте, модель не сможет извлекать ключевые слова из тестовых текстов.

Теперь выполним проверку RKEA на текстах большого объема. Для этого воспользуемся датасетом Krapivin2009, содержащим 2304 научных работы, посвященных информатике. Датасет уже содержит в себе набор ключевых фраз к каждой работе. Обучение модели так же будет происходить с использованием модельных примеров, статей из Reuters и научных работ из датасета.

Для тестирования были взяты две разные статьи – первая из датасета Krapivin2009, вторая – статья, посвященная клиентскому оттоку

Таблица 3. Исследование извлечения ключевых слов из научных статей

Входные данные				Выходные данные				
№ текста	n, объем текста	N_{tr}, кол-во тренировочных текстов	Обучение модели	t_o, время обучения, с	$T_и$, время извлечения, с	mem_o, память при обучении	$mem_и$, память при извлечении	lev, сходство эксп. и теор. ключ. слов
1	45151	10	модель	0.084	0.055	768 B	3.23 kB	0.2229
		50		0.451	0.074	-2.56 kB	3.3 kB	0.23636
		10	Reuters	0.032	0.055	5.97 kB	4.06 kB	0.25754
		50		0.254	0.071	22.8 kB	3.32 kB	0.32687
		10	Krapivin	1.61	0.117	5.76 kB	3.54 kB	0.39649
		50		8.085	0.214	22.8 kB	3.23 kB	0.27961
		100		16.589	0.378	46.7 kB	3.76 kB	0.49102
		500		89.367	1.65	4.93 kB	4.23 kB	0.45268
2	22535	10	модель	0.088	0.036	5.97 kB	3.3 kB	0.3087
		50		0.422	0.043	768 B	3.3 kB	0.30839
		10	Reuters	0.036	0.034	6.18 kB	3.3 kB	0.38656
		50		0.155	0.038	22.6 kB	3.44 kB	0.3937
		10	Krapivin	1.784	0.076	6.18 kB	3.3 kB	0.44187
		50		7.826	0.169	22.8 kB	3.23 kB	0.49356
		100		15.384	0.257	43.6 kB	3.91 kB	0.49356
		500		91.884	1.967	113 kB	4.08 kB	0.40287

5. Сравнение извлечения ключевых слов в языке Python с языком R.

В языке Python имеется множество библиотек для извлечения ключевых слов, но наиболее простой и эффективной, а вследствие и популярной является библиотека KeyBERT. В библиотеке используется коэффициент Отиаи (косинусный коэффициент) для классификации ключевых слов.

По сравнению с библиотекой RKEA эта библиотека является более гибкой и позволяет указывать из какого количества слов могут состоять извлекаемые фразы, первые N ключевых фраз и пр.

Модели в библиотеке не требуется обучение, поэтому извлекать слова можно без предварительного ручного определения ключевых слов.

Входные данные		Выходные данные			
Текст	n, объем текста	$T_и$, время извлечения, с	$mem_и$, память при извлечении	Сходство результата KeyBERT и теоретических слов	Сходство результата RKEA и теоретических слов
Статья из Reuters	646	2,897	185.73 кБ	0.54967	0.40865
Статья из Krapivin	45151	7,975	324.71 кБ	0.571	0.45268
Customer Churn	22535	6,246	315.26 кБ	0.44679	0.30839

Как видно, библиотека KeyBERT показала большее сходство с теоретическими ключевыми фразами, чем RKEA, однако у этой библиотеки существенно больше времени уходит на извлечение ключевых фраз. Также стоит учитывать, что в RKEA требуется время на обучение модели, в то время как в KeyBERT обучение не требуется и этот процесс не занимает время.

Выводы

В ходе выполнения практической работы были разработаны программы, выполняющие извлечение ключевых слов из текстов с помощью библиотек RKEA (язык R) и KeyBERT (Python).

Тестирование программ проводилось на модельных примерах путем моделирования текста набором случайных слов, а также на реальных примерах в виде кратких выдержек статей из Reuters, научных статей из датасета Krapivin2009 и статьи о клиентском оттоке.

RKEA наиболее эффективно показала себя на анализе объемных текстов с обучением модели на примере объемных текстов с похожей тематикой. Учитывая возможность подключения автоматизированного словаря со специфическими терминами эта библиотека способна с высокой точностью извлекать ключевые слова из научные статьи большого объема.