# STA323

Big Data Analysis Software and Application (Hadoop or Spark)

Report on Project 1

12112627 李乐平

**Question 1.** Security Data Analysis.

MACCDC 2012 provides operational data of managing and protecting an existing network infrastructure.

(1). Create two Spark dataframes ( `df_http, df_dns` ) from the files `http.log.gz, dns.log.gz` in folders `00` to `05` (six folders in total). Convert the `ts` column to Timestamp data type. Create two temp view named `http_log` and `dns_log`. [2 points]

Answer:

首先导入数据，其中需要把状态码列手动转为整型。

```
file_paths_http = [
    "./data/task1/00/http.log.gz", "./data/task1/01/http.log.gz", "./data/task1/02/http.log.gz",
    "./data/task1/03/http.log.gz", "./data/task1/04/http.log.gz", "./data/task1/05/http.log.gz"
]
file_paths_dns = [
    "./data/task1/00/dns.log.gz", "./data/task1/01/dns.log.gz", "./data/task1/02/dns.log.gz",
    "./data/task1/03/dns.log.gz", "./data/task1/04/dns.log.gz", "./data/task1/05/dns.log.gz"
]
df_http = spark.read.json(file_paths_http).withColumn("status_code", col("status_code").cast("int"))
df_dns = spark.read.json(file_paths_dns)
```

转换 ts 类型，然后创建视图。

```
df_http = df_http.withColumn("ts", to_timestamp(df_http["ts"]))
df_dns = df_dns.withColumn("ts", to_timestamp(df_dns["ts"]))

df_http.createOrReplaceTempView("http_log")
df_dns.createOrReplaceTempView("dns_log")
```

```
+--------------+--------------+---------+--------------+---------+------+-------------+--------------------+--------------+------+---
----+--------------------+----------------+-------------+--------------------+--------------+----------------+-----------+-------------
----+----+----------+--------------------+----------------+-----------------+--------------------+--------------+--------+------+
|           host|      id.orig_h|id.orig_p|      id.resp_h|id.resp_p|method|orig_filenames|          orig_fuids|orig_mime_types|origin|pro
xied|            referrer|request_body_len|resp_filenames|          resp_fuids|resp_mime_types|response_body_len|status_code|       status
_msg|tags|trans_depth|                  ts|              uid|                uri|          user_agent|username|version|
+--------------+--------------+---------+--------------+---------+------+-------------+--------------------+--------------+------+---
----+--------------------+----------------+-------------+--------------------+--------------+----------------+-----------+-------------
----+----+----------+--------------------+----------------+-----------------+--------------------+--------------+--------+------+
| 192.168.202.78| 192.168.204.70|    46197| 192.168.202.78|       80|  POST|          NULL|[FPp8eT1dDgDcIzdRT7]|   [text/plain]|  NULL|
NULL|http://192.168.20...|            4490|         NULL|                NULL|          NULL|               0|        302|            F
ound|  []|          1|2012-03-16 15:12:...| C1odva1GOl62Mubql|           /doku.php|Mozilla/5.0 (X11;...|    NULL|    1.1|
|192.168.205.253|192.168.202.107|     1191|192.168.205.253|       80|   GET|          NULL|                NULL|          NULL|  NULL|
NULL|                NULL|               0|         NULL| [FLeXV6ZoEWi12BLol]|    [text/html]|              56|        301|Moved Permane
ntly|  []|          1|2012-03-16 15:12:...| CQLw22eZ8JIDpJcik|/sdk/vimService?wsdl|                NULL|    NULL|    1.1|
|   127.0.0.1:80|192.168.202.102|     1112| 192.168.22.152|       80|   GET|          NULL|                NULL|          NULL|  NULL|
NULL|                NULL|               0|         NULL|[FVOd8i2QjtprIkVwX1]|    [text/html]|              44|        200|
OK|  []|          1|2012-03-16 15:12:...|CKaLaf1peiUU3kmTV8|         /index.html|                NULL|    NULL|    1.1|
| 192.168.202.78| 192.168.204.70|    46198| 192.168.202.78|       80|   GET|          NULL|                NULL|          NULL|  NULL|
NULL|http://192.168.20...|               0|         NULL|[FImEgi2hnmej8PAoui]|    [text/html]|           22249|        200|
OK|  []|          1|2012-03-16 15:12:...|CtEYyv1YQSocDA5oY6| /doku.php?id=start&|Mozilla/5.0 (X11;...|    NULL|    1.1|
| 192.168.202.78| 192.168.204.70|    46200| 192.168.202.78|       80|   GET|          NULL|                NULL|          NULL|  NULL|
NULL|http://192.168.20...|               0|         NULL|                NULL|          NULL|               0|        304|     Not Modi
fied|  []|          1|2012-03-16 15:12:...|CwrlVC14uolzZt7dPc|/lib/tpl/default/...|Mozilla/5.0 (X11;...|    NULL|    1.1|
+--------------+--------------+---------+--------------+---------+------+-------------+--------------------+--------------+------+---
----+--------------------+----------------+-------------+--------------------+--------------+----------------+-----------+-------------
----+----+----------+--------------------+----------------+-----------------+--------------------+--------------+--------+------+
only showing top 5 rows
```

```
+-----+-----+----+-----+----+---+-------+-------------+---------+-------------+---------+-----+------+-----------+-----+----------+------
------------+-----+----------+--------+----+--------+-------------------+-----------------+
|   AA|   RA|  RD|   TC|TTLs|  Z|answers|     id.orig_h|id.orig_p|    id.resp_h|id.resp_p|proto|qclass|qclass_name|qtype|qtype_name|
query|rcode|rcode_name|rejected| rtt|trans_id|                 ts|              uid|
+-----+-----+----+-----+----+---+-------+-------------+---------+-------------+---------+-----+------+-----------+-----+----------+------
------------+-----+----------+--------+----+--------+-------------------+-----------------+
|false|false|true|false|NULL|  0|   NULL|192.168.204.70|    38795|192.168.207.4|       53|  tcp|     1| C_INTERNET|   28|      AAAA|creati
vecommons.org|    3|  NXDOMAIN|   false|NULL|   50106|2012-03-16 15:12:...| C1odva1GOl62Mubql|
|false|false|true|false|NULL|  0|   NULL|192.168.204.70|    53918|192.168.207.4|       53|  udp|     1| C_INTERNET|   28|      AAAA|   ww
w.dokuwiki.org|    3|  NXDOMAIN|   false|NULL|   52251|2012-03-16 15:12:...| CQLw22eZ8JIDpJcik|
|false|false|true|false|NULL|  0|   NULL|192.168.204.70|    45058|192.168.207.4|       53|  udp|     1| C_INTERNET|   28|      AAAA|creati
vecommons.org|    3|  NXDOMAIN|   false|NULL|   61533|2012-03-16 15:12:...|CKaLaf1peiUU3kmTV8|
|false|false|true|false|NULL|  0|   NULL|192.168.204.70|    56979|192.168.207.4|       53|  udp|     1| C_INTERNET|   28|      AAAA|
www.php.net|    3|  NXDOMAIN|   false|NULL|   37509|2012-03-16 15:12:...|CtEYyv1YQSocDA5oY6|
|false|false|true|false|NULL|  0|   NULL|192.168.204.70|    35954|192.168.207.4|       53|  udp|     1| C_INTERNET|   28|      AAAA|   ww
w.dokuwiki.org|    3|  NXDOMAIN|   false|NULL|   17572|2012-03-16 15:12:...|CwrlVC14uolzZt7dPc|
+-----+-----+----+-----+----+---+-------+-------------+---------+-------------+---------+-----+------+-----------+-----+----------+------
------------+-----+----------+--------+----+--------+-------------------+-----------------+
only showing top 5 rows
```

(2). With the `http_log` data, filter the rows where the status code is `200` and method is `GET`, sort in a descending order according to the accessed count of the `uri`. Use Spark SQL API and Spark dataframe, seperately. [2 points]

Answer:

使用 SQL 查询即可。

```
http_log_filtered = spark.sql("""
    select
        uri,
        count(*) as accessed_count
    from
        http_log
    where
        status_code = 200 and method = 'GET'
    group by
        uri
    order by
        accessed_count desc
""")
```

```
+--------------------+--------------+
|                 uri|accessed_count|
+--------------------+--------------+
|                   /|          9475|
|/admin/config.php...|           556|
|  /main.php?logout=1|           194|
|/top.php?stuff=15...|           191|
|            /top.php|           179|
+--------------------+--------------+
```

(3). Use Spark SQL to join the `http_log` and `dns_log` tables by `uid`, and calculate the percentage of `proto=tcp` for each `uri` group found in task (2). [3 points]

Answer:

使用前一问的 SQL 作为子查询进行联合查找即可，结果共 389 条。

```
result_df_sql = spark.sql("""
    select
        h.uri,
        100.0 * sum(case when proto = "tcp" then 1 else 0 end) / count(*) as tcp_percentage
    from
        (
            select
                uri
            from
                http_log
            where
                status_code = 200
            and
                method = 'GET'
            group by
                uri
        ) as selected_uris,
```

```
        http_log h,
        dns_log d
    where
        h.uid = d.uid
    and
        h.uri = selected_uris.uri
    group by
        h.uri
    order by
        tcp_percentage desc
""")
```

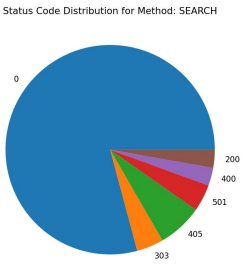
```
+-----------------------------------------------------------------------------------------------+------------------+
|uri                                                                                            |tcp_percentage    |
+-----------------------------------------------------------------------------------------------+------------------+
|/lib/exe/indexer.php?id=start&1331903710                                                       |100.00000000000000|
|/smconf.nsf                                                                                    |100.00000000000000|
|/classes/                                                                                      |100.00000000000000|
|/cgilib/                                                                                       |100.00000000000000|
|/_private/                                                                                     |100.00000000000000|
```

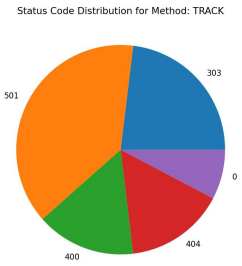
(4). Use Spark DataFrame to calculate the percentage of different method in the `http_log`. Also display the pie chart of different status code for each method. [2 points]

Answer:

由于输出图片较多，其余图片将以附件的形式上传。其中，缺失值被置 `0`。

```
method_counts = df_http.groupBy("method").agg(count("*").alias("count"))
total_count = method_counts.selectExpr("sum(count) as total").collect()[0]["total"]
method_percentages = method_counts.withColumn("percentage", (col("count") / total_count) * 100)
pie_data = method_percentages.toPandas()
for method in pie_data["method"]:
    method_df = df_http.filter(df_http["method"] == method)
    status_counts = method_df.groupBy("status_code").agg(count("*").alias("count"))
    status_data = status_counts.toPandas()
    status_data["status_code"] = status_data["status_code"].fillna(0).astype(int)
    plt.figure(figsize=(8, 6), dpi = 160)
    plt.pie(status_data["count"], labels=status_data["status_code"])
    plt.title(f"Status Code Distribution for Method: {method}")
    plt.savefig(f"./output/task1/pie_chart_{method}.png")
    plt.close()
```




## Question 2. Document Analysis

Paul Graham is an English computer scientist, essayist, entrepreneur, investor, and author. He is best known for his work on the programming language Lisp, co-founding the influential startup accelerator and seed capital firm Y Combinator, and Hacker News.

(1). Crawl the articles of Paul Graham and store the text of the articles in folder `paul_articles`. Each article is located in a seperate `.txt` file. You can use Scrapy or any other tools you like. You can refer to the official guidelines of Scrapy here. [3 points]

Answer: 、

本题代码见 `task2_make_txt.ipynb`。结果被保存在`./data/task2/paul_articles/`中。

首先访问文章列表获得各文章对应的 `uri`。

```
base_url = "https://paulgraham.com/"
base_uri = "articles.html"
base_response = requests.get(url = base_url + base_uri)
base_soup = BeautifulSoup(base_response.content, "html.parser")
hrefs = [a["href"] for a in base_soup.find_all('a') if a["href"].endswith("html")][1:]
```

然后访问爬取的 `uri` 并从中提取文章主体内容，其中需要注意自然段的分割。代码中的做法是，先提取换行标记`<br>`和`<p>`并替换为自定义的标记`"^^^"`，使之之后不会在 `get_text` 函数调用时被忽略。然后将原有的换行符替换为空格，因其并不能代表自然段的分割。再将标记`"^^^"`换为换行符，并将多余的空格删除，即可得到以正常自然段分隔的文章。

```
for uri in hrefs:
    print(base_url + uri)
    response = requests.get(url = base_url + uri)
    soup = BeautifulSoup(response.content, "html.parser")
    font_tag = soup.find('font', size="2", face="verdana")
    article_font = soup.find('font')
    if article_font:
        for br in article_font.find_all("br"):
            br.replace_with("^^^")
        for p in article_font.find_all("p"):
            p.replace_with("^^^")
        article_content = article_font.get_text()
        article_content = article_content.replace("\n", " ")
        article_content = article_content.replace("^^^", "\n")
        article_content = article_content.replace("  ", " ")
        article_content = article_content.replace("\n\n", "\n")
        print(f"Done with {uri}")
        with open("./data/task2/paul_articles" + uri + ".txt", 'w', encoding='utf-8') as file:
            file.write(article_content)
        time.sleep(random.random() * 3)
    else:
        print("Article content not found.")
        break
```

(2). Create a dataframe by reading from these `.txt` files with `pyspark`. Each row only contains the sentences in one paragraph. Select paragraphs that are related to suggestions of career planning. You can read some examples to determine some key phrases or regrex expressions for filtering. Store all the filtered paragraphs in a parquet file `career_suggestions.parquet`. (This is an open task that different answers are allowed) [3 points]

Answer:

本题代码见 `task2_filter_paragraph.ipynb`。

首先读入目标目录中的`.txt` 文件。

```
text_df = spark.read.text("./data/task2/paul_articles/*.txt").filter(col("value") != '')
```

```
--------------------------------------------------------------------------------+
|value
|
+-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
|February 2021
|
|Before college the two main things I worked on, outside of school, were writing and programming. I didn't write essays. I wrote what begi
nning writers were supposed to write then, and probably still are: short stories. My stories were awful. They had hardly any plot, just ch
aracters with strong feelings, which I imagined made them deep.
|
|The first programs I tried writing were on the IBM 1401 that our school district used for what was then called "data processing." This wa
s in 9th grade, so I was 13 or 14. The school district's 1401 happened to be in the basement of our junior high school, and my friend Rich
Draves and I got permission to use it. It was like a mini Bond villain's lair down there, with all these alien-looking machines — CPU, dis
k drives, printer, card reader — sitting up on a raised floor under bright fluorescent lights.
|
|The language we used was an early version of Fortran. You had to type programs on punch cards, then stack them in the card reader and pre
ss a button to load the program into memory and run it. The result would ordinarily be to print something on the spectacularly loud printe
r.
|
|I was puzzled by the 1401. I couldn't figure out what to do with it. And in retrospect there's not much I could have done with it. The on
ly form of input to programs was data stored on punched cards, and I didn't have any data stored on punched cards. The only other option w
as to do things that didn't rely on any input, like calculate approximations of pi, but I didn't know enough math to do anything interesti
ng of that type. So I'm not surprised I can't remember any programs I wrote, because they can't have done much. My clearest memory is of t
he moment I learned it was possible for programs not to terminate, when one of mine didn't. On a machine without time-sharing, this was a
social as well as a technical error, as the data center manager's expression made clear.|
+-------------------------------------------------------------------------------------------------------------------------------------------------------
```

然后筛选出与职业建议相关的段落并保存为`.parquet`。

```
career_keywords = [
    "career", "job", "profession", "employment", "work", "occupation", "vocation", "future", "plan",
    "skill", "qualif", "achieve", "passion", "suggest"
]
regex_pattern = "|".join([f"(?=.*{keyword})" for keyword in career_keywords])
filtered_df = text_df.filter(
    col("value").rlike(regex_pattern)
)

filtered_df.write.parquet("./output/task2/career_suggestions.parquet", mode = "overwrite")
filtered_df.show(5, truncate = False)
```

(3). Extract noun phrases of all articles with Spark user-defined-functions and count their frequencies. You can use the `Spacy` Package. Plot the word cloud map with `wordcloud` package for the noun phrases which have the top 40~50 highest frequencies (including both end). [4 points]

Answer:

首先使用 `noun_chunks` 提取出名词片段。为使名词短语有意义，其应由 2 个以上的单词构成名词属性的成分，并且以名词作为开头，据此以首个单词为名词作为筛选条件。

```
nlp = spacy.load("en_core_web_sm")

def extract_noun_phrases(text):
    doc = nlp(text)
    return [chunk.text.lower() for chunk in doc.noun_chunks if chunk[0].pos_ == 'NOUN']

extract_noun_phrases_udf = udf(extract_noun_phrases, ArrayType(StringType()))
noun_phrases_df = text_df.select(explode(extract_noun_phrases_udf("value")).alias("noun_phrases"))
noun_phrases_df = noun_phrases_df.filter(~trim(col("noun_phrases")).rlike("^\\S+$"))
noun_phrases_df =
noun_phrases_df.filter(~(col("noun_phrases").rlike(r'\b(?:a|the|an|one|this|that|your|their)\b')))
```

```
+-------------------------+
|noun_phrases             |
+-------------------------+
|beginning writers        |
|disk drives              |
|card reader              |
|punch cards              |
|calculate approximations|
+-------------------------+
```

然后根据各短语出现的频数进行排序，取前 40-50 个绘制词云图。

```
noun_phrase_freq = noun_phrases_df.groupBy("noun_phrases").count().orderBy("count", ascending=False)
top_noun_phrases = noun_phrase_freq.select("noun_phrases", "count").collect()[39:50]

word_freq_dict = {row["noun_phrases"]: row["count"] for row in top_noun_phrases}

wordcloud = WordCloud(width=800, height=400,
background_color="white").generate_from_frequencies(word_freq_dict)

plt.figure(figsize=(10, 5), dpi = 160)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```