به نام خدا

دانشگاه شهید بهشتی تهران

دانشکده برق و مهندسی کامپیوتر

تز ارشد تشخیص ناهنجاری

بخش تجزیه لاگ HDFS

توسط

مصطفی جعفری

40144303

هدف ما در این بخش، دریافت لاگ خام HDFS و تبدیل آن به بردارهای متوالی(sequence vectors) است تا بتوانیم به مدل DL وارد کنیم که مراحل زیر را دارد، توجه شود که بردار شمارشی(quantitative vactor) به راحتی از این بردار استخراج می شود و ایجاد بردار معنایی(semantic vector) در گزارش بعدی مفصل بررسی می شود.

1. دریافت فایل خام لاگ های HDFS و لیبل ها
2. آپلود یک log parser که در این جا از Drain به دلیل نتایج دقیق تر استفاده می کنیم ولی می توان به سادگی هر مدل دیگر استفاده کرد.
3. تبدیل لاگ خام به لاگ ساختاریافته توسط log parser و همچنین استخراج قالب ها(templates)
4. با توجه به این که در این دیتاست خاص، لاگ ها دارای شناسه(identity) خاص هستند، دسته بندی لاگ های هر بلوک خاص(لاگ هایی که در آن بلوک یا گره خاص ثبت شده اند)
5. اضافه کردن لیبل به لاگ هر بلوک
6. تقسیم دیتاست با توجه به روش تشخیص ناهنجاری به دو بخش train و test

کد کامل این گزارش :

https://github.com/mostafa-ja/Anomaly-detection/blob/main/A(parsing).ipynb

مرحله 1

سایت https://github.com/logpai/loghub ، تقریبا تمامی لاگ های متن باز استاندارد موجود که در مقالات استفاده شده را جمع آوری کرده است که شامل دیتاست HDFS هم می باشد فقط به دلیل حجم بالای لاگ ها که قابلیت آپلود در گیت هاب را ندارد به وبسایت دیگری(zenodo.org) برای دانلود دیتاست کامل ارجاع میدهد، همچنین برای هر دیتاست، 2k لاگ به عنوان نمونه قرار داده است. در این مرحله از آدرس https://zenodo.org/record/3227177 داده HDFS_1.tar.gz را دانلود میکنیم.

به عنوان مثال سطر اول لاگ خام به صورت زیر است

```
081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010
```

که فرمت لاگ به صورت (<Date> <Time> <Pid> <Level> <Component>: <Content>) است، که در قسمت Content ما شناسه بلوک (blk_38865049064139660) را می بینیم. در حالت ساختار یافته، این خط لاگ تبدیل به شکل زیر میشود، که بخش های مختلف جدا شده، قالب خاص لاگ بدون پارامترها ایجاد شده(EventTemplate) و لیست پارامترها(ParameterList) جدا در ستون آخر اضافه شده است.

Date	Time	Pid	Level	Component	Content	EventId	EventTemplate	ParameterList
81109	203518	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	bbb51b95	Receiving block <*> src: /<*> dest: /<*>	['blk_-1608999687919862906', '10.250.19.102:54...

مرحله 2

دریافت drain از ریپازیتوری https://github.com/logpai/logparser که مجموعه ای از log parser های متن باز رو جمع آوری کرده است. البته این فایل ایراد جزئی در جداسازی پارامترها داشت که باعث می شد لیست پارمترها را ذخیره نکند که با تغییر جزئی، این مشکل رفع شد و فایل اصلاح شده در گیت هاب در آدرس https://github.com/mostafa-ja/Anomaly-detection/blob/main/datasets/Drain.py ذخیره شد.

مرحله 3

در کد زیر، تنظیمات اولیه برای تجزیه لاگ توسط drain را مشاهده می کنیم

```
input_dir  = '/content/'  # The input directory of log file
output_dir = 'Drain_result/'  # The output directory of parsing results
log_file   = 'HDFS.log'  # The input log file name
log_format = '<Date> <Time> <Pid> <Level> <Component>: <Content>'  # HDFS log format
# Regular expression list for optional preprocessing (default: [])

regex      = [
    r'blk_(|-)[0-9]+' , # block id
    r'(/|)([0-9]+\.){3}[0-9]+(:[0-9]+|)(:|)', # IP
    r'(?<=[^A-Za-z0-9])(\-?\+?\d+)(?=[^A-Za-z0-9])|[0-9]+$', # Numbers
]
st         = 0.5  # Similarity threshold
depth      = 4  # Depth of all leaf nodes

#keep_para=True >> we will also have ParameterList column contains parameters which are removed
parser = Drain.LogParser(log_format, indir=input_dir, outdir=output_dir,  depth=depth, st=st, rex=regex, keep_para=True)
parser.parse(log_file)
```

```
081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_38865049064139660 terminating
```

date time PID level component content

Log_format : این که هنگام ساختار دادن به هر سطر متن، به چه تعداد ستون تقسیم کند همچنین نام ستون ها را مشخص میکنیم، به عنوان مثال در این کد نشان میدهیم که به 6 ستون با توجه به فاصله ها، تقسیم کند اگر مثلا 5 ستون تعریف کنیم، به ازای فاصله، ستون ها را جدا میکند و مجبور میشود level و component را یکی کند چون علامت(:) باعث جدا شدن قسمت آخر از بقیه متن شده است.

St و depth : مشخصات تجزیه به صورت درخت (این روش از درخت برای دسته بندی و یافتن قالبها استفاده میکند)را مشخص میکند که آستانه شباهت و عمق درخت را مشخص میکند.

Reg : نشان می دهیم چه قسمتهایی از متن لاگ در ایجاد قالب ها حذف شود که اعداد، آی پی و شناسه بلوک می باشد، که پارامترهای متن هستند، که بعدا از شناسه بلوک برای تفکیک لاگ ها استفاده می شود و بقیه پارامتر ها به صورت لیست جدا ذخیره می شوند.

keep_para : در صورت true بودن، لیست پارامترهای هر خط لاگ را به صورت جدا در ستون ParameterList ذخیره می کند.

مرحله 4

با اجرای کد بالا، لاگ ساختاریافته (struct_log) ایجاد می شود که حجم بسیار بالایی دارد، توجه شود که کل لاگ بیش از 11 میلیون خط می باشد.

در شکل زیر قالب های متمایز را مشاهده می کنیم که با توجه به تنظیمات اولیه log parser، 48 قالب ایجاد شده است که log parser هر نوع قالب را با کد طولانی خاصی نام گذاری کرده است.

```
#uniques: unique templates of the 'EventId' column
#labels : the mapping dictionary of unique items to Numeric Representation
labels, uniques = pd.factorize(struct_log['EventId'])

print("Unique templates : \n", uniques)
print("Numeric Representation : \n", labels)
```

```
Unique templates :
 Index(['bbb51b95', '3d91fa85', 'd38aa58d', '46003790', '5d5de21c', '44614d71',
       '9c784e29', '75627efd', '54e5f6b4', '4dec0816', '728076ac', '40651754',
       '32777b38', 'ace40671', 'e817fa45', 'bcc910df', 'dba996ef', '0567184d',
       '69bca6e5', 'd013b7a3', 'c294d20f', 'a5478aff', 'f7c33085', 'f266840a',
       '15b484af', '9621139e', 'd62e5638', '1061e5d9', '2ecc047e', '4610d0f1',
       '1a718242', 'd63ef163', '2e68ccc3', '8f2bc724', 'a333d363', 'e024fa48',
       '25382c88', '20317105', '06d16156', '78915d3a', '13eb7010', 'f79898ae',
       '124068c6', '625a2a34', 'b65fc512', 'fcd37a6d', '559305d8', 'fcf2c482'],
      dtype='object')
Numeric Representation :
 [ 0  1  0 ... 12 12 12]
```

می توان به هر قالب یک کد ساده که از صفر شروع می شود نسبت داد(ستون EventId) تا در مراحل بعد و در ایجاد بردار و تحلیل لاگ، کار آسان تر شود

```
# Use pd.factorize() to encode the 'EventId' column
struct_log['EventId'] = labels
struct_log.head(3)
```

LineId	Date	Time	Pid	Level	Component	Content	EventId	EventTemplate	ParameterList
1	81109	203518	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	0	Receiving block <*> src: /<*> dest: /<*>	['blk_-1608999687919862906', '10.250.19.102:54...
2	81109	203518	35	INFO	dfs.FSNamesystem	BLOCK* NameSystem.allocateBlock: /mnt/hadoop/m...	1	BLOCK* NameSystem.allocateBlock: <*> <*>	['/mnt/hadoop/mapred/system/job_200811092030_0...
3	81109	203519	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	0	Receiving block <*> src: /<*> dest: /<*>	['blk_-1608999687919862906', '10.250.10.6:4052...

سپس لاگ های مربوط به هر شناسه بلوک(BlockId) را جدا میکنیم و ستون EventSequence را ایجاد می کنیم، توجه شود که هر لاگ با قالب اصلی آن(یک عدد بین صفر تا 47) نشان داده می شود.

```
data_dict = OrderedDict()  # to search faster
for idx, row in struct_log.iterrows():
    blkId_list = re.findall(r'(blk_-?\d+)', row['Content'])
    blkId_set = set(blkId_list)
    for blk_Id in blkId_set:
        if not blk_Id in data_dict:
            data_dict[blk_Id] = []
        data_dict[blk_Id].append(row['EventId'])
data_df = pd.DataFrame(list(data_dict.items()), columns=['BlockId', 'EventSequence'])
data_df.to_csv("HDFS_sequence.csv",index=None)

data_df.head(3)
```

	BlockId	EventSequence
0	blk_-1608999687919862906	[0, 1, 0, 0, 2, 2, 3, 3, 2, 3, 4, 4, 4, 5, 0, ...
1	blk_7503483334202473044	[0, 0, 1, 0, 2, 3, 2, 3, 2, 3, 4, 4, 4, 9, 12,...
2	blk_-3544583377289625738	[0, 1, 0, 0, 2, 3, 2, 3, 2, 3, 9, 4, 4, 4, 9, ...

مرحله 5

در شکل زیر برچسب داده ها(anomaly_label) را مشاهده می کنیم ، برچسب های موجود در دیتاست با توجه به شناسه بلوک طبقه بندی شده اند و لاگ هر بلوک براساس شناسه آن، برچسب normal و یا anomaly را دارد.

```
labels = pd.read_csv('/content/drive/MyDrive/HDFS/anomaly_label.csv')
labels.head()
```

	BlockId	Label
0	blk_-1608999687919862906	Normal
1	blk_7503483334202473044	Normal
2	blk_-3544583377289625738	Anomaly
3	blk_-9073992586687739851	Normal
4	blk_7854771516489510256	Normal

حال برچسب مربوط به هر شناسه بلوک را، به داده بدست آمده از مرحله 4 مطابق شکل زیر اضافه می کنیم

```
# Merge the labels with the data_df DataFrame
data_df = data_df.merge(labels, on='BlockId', how='left')
data_df.head(3)
```

	BlockId	EventSequence	Label
0	blk_-1608999687919862906	[0, 1, 0, 0, 2, 2, 3, 3, 2, 3, 4, 4, 4, 5, 0, ...	Normal
1	blk_7503483334202473044	[0, 0, 1, 0, 2, 3, 2, 3, 2, 3, 4, 4, 4, 9, 12,...	Normal
2	blk_-3544583377289625738	[0, 1, 0, 0, 2, 3, 2, 3, 2, 3, 9, 4, 4, 4, 9, ...	Anomaly

مرحله 6

برای تقسیم دیتاست به دو قسمت train و test باید با توجه به روش تشخیص ناهنجاری صورت گیرد، برای روش semi-supervised باید قسمت train فقط شامل لاگ نورمال باشد و test شامل داده نورمال و هم غیرنورمال باشد در حالی که در روش supervised، به هر دو نوع داده در هر دو بخش نیاز داریم. پس در این بخش ما تنها داده را با توجه به برچسب آن به دو بخش نورمال و غیرنورمال تقسیم می کنیم و در مراحل بعد با توجه به روش تشخیص و نسبت داده train و test ، داده ها را جدا می کنیم.

```
# Splitting the dataset into train and test
hdfs_sequence_normal = data_df[data_df['Label'] == 'Normal']

hdfs_sequence_abnormal = data_df[data_df['Label'] != 'Normal']

# Print the lengths of train and test datasets
print("Normal Dataset Length:", len(hdfs_sequence_normal))
print("Abnormal Test Dataset Length:", len(hdfs_sequence_abnormal))
```

```
Normal Dataset Length: 558223
Abnormal Test Dataset Length: 16838
```

در شکل زیر داده های مربوط به هر بخش را می بینیم

```
hdfs_sequence_normal.head(3)
```

	BlockId	EventSequence	Label
0	blk_-1608999687919862906	[0, 1, 0, 0, 2, 2, 3, 3, 2, 3, 4, 4, 4, 5, 0, ...	Normal
1	blk_7503483334202473044	[0, 0, 1, 0, 2, 3, 2, 3, 2, 3, 4, 4, 4, 9, 12,...	Normal
3	blk_-9073992586687739851	[0, 1, 0, 0, 2, 3, 2, 3, 2, 3, 4, 4, 4, 12, 12...	Normal

```
hdfs_sequence_abnormal.head(3)
```

	BlockId	EventSequence	Label
2	blk_-3544583377289625738	[0, 1, 0, 0, 2, 3, 2, 3, 2, 3, 9, 4, 4, 4, 9, ...	Anomaly
15	blk_-8531310335568756456	[0, 1, 0, 0, 2, 3, 2, 3, 2, 3, 4, 4, 4, 12, 12...	Anomaly
23	blk_3947106522258141922	[0, 0, 1, 0, 2, 3, 2, 3, 4, 4, 2, 3, 4, 12, 12...	Anomaly

سپس این دو جدول hdfs_sequence_normal و hdfs_sequence_abnormal را به صورت متن ذخیره می کنیم که برای هر جدول در هر خط، توالی قالب های مربوط به یک بلوک خاص ذخیره می شود، زیرا برای تحلیل، ما نیازی به شناسه بلوک نداریم، تنها برای ما تفکیک لاگ های هر بلوک مهم می باشد .

```
# save hdfs_sequence file in google drive

with open('/content/drive/MyDrive/HDFS/structured_hdfs/hdfs_sequence_normal', 'w') as log_file:
  for _ , row in hdfs_sequence_normal.iterrows():
      log_entry  = ' '.join(str(token) for token in row['EventSequence'])
      log_file.write(log_entry + '\n')

with open('/content/drive/MyDrive/HDFS/structured_hdfs/hdfs_sequence_abnormal', 'w') as log_file:
  for _ , row in hdfs_sequence_abnormal.iterrows():
      log_entry  = ' '.join(str(token) for token in row['EventSequence'])
      log_file.write(log_entry + '\n')
```

در مراحل بعد می توان این داده را باتوجه به روش تحلیل، با نسبت خاص جدا کرد.

در شکل زیر چند خط از فایل لاگ نورمال ایجاد شده را مشاهده می کنیم که هر خط مربوط به لاگ یک شناسه بلوک خاص می باشد.

```
22 5 5 5 26 26 26 11 9 11 9 11 9 23 23 23 21 21 21

5 5 5 22 11 9 11 9 11 9 26 26 26 4 3 4 23 23 23 21 21 21

5 22 5 5 11 9 11 9 9 26 26 11 26 3 4 3 2 3 4 3 23 23 23 21 21 21

5 5 22 5 26 11 9 11 9 11 9 26 26 23 23 23 21 21 21

5 22 5 5 26 11 9 11 9 11 9 26 26 4 4 3 23 23 23 21 21 21
```

نکات مهم

تنظیمات اولیه log parser باعث ایجاد تعداد قالب های تفکیک شده متفاوت می شود، مثلا با تنظیمات ایجاد شده ما، 48 قالب متمایز ایجاد می شود که در حالت تفکیک بیش از حد ممکن است تعدادی از قالب ها بسیار شبیه به هم باشند و بی دلیل تفکیک شده اند یا بالعکس، که تاثیرعمیقی به نتایج نهایی تشخیص ناهنجاری می گذارد. در صورت استفاده از بردار معنایی(semantic vector) به جای قالب های اصلی در ورودی مدل، اهمیت شباهت قالب های ایجاد شده بسیار کم می شود زیرا بردار معنایی آن قالب ها بسیار مشابه هم می شوند تنها در پیش بینی لاگ(قالب) بعدی ممکن است با مشکل مواجه شویم که برای لاگ بعدی، تعدادی قالب با احتمال بسیار نزدیک به هم، ممکن است داشته باشیم که می تواند درد سر ساز شود و باعث کاهش دقت مدل شود، همچنین زمانی که از بردار توالی(sequence vector) یا بردار شمارشی(quantitative vector) استفاده می کنیم این تاثیر محسوس تر است که می تواند باعث نتایج کاملا متفاوتی شود، مخصوصا وقتی که می خواهیم قالب بعدی را پیش بینی کنیم.

در ریپازیتوری https://github.com/logpai/logparser که به تحلیل log parser ها پرداخته است و همچنین مجموع دیتاست لاگ ها(مرجع اصلی لاگ های متن باز) نیز مربوط به همین کاربر است، تعداد قالب های اصلی دیتاست HDFS (ground truth) که به صورت دستی تفکیک کرده است و معیار مقایسه قرار داده است 30 می باشد ولی با تنظیمات اولیه قرار داده در ریپازیتوری و اجرای log parser، این تعداد به 48 می رسد، دقیقا عددی که مربوط به مقاله جدید و مطرح (Log-based Anomaly Detection with Deep Learning: How Far Are We) می باشد که لاگ ها را با استفاده از Drain تجزیه کرده است و در آدرس زیر است

https://github.com/LogIntelligence/LogADEmpirical

همچنین در دو ریپازیتوری معروف زیر در تشخیص ناهنجاری لاگ HDFS از لاگ ساختار یافته استفاده کرده اند که تعداد آن 28 می باشد و اشاره ای به مرجع آن ندارند.

https://github.com/donglee-afar/logdeep

https://github.com/wuyifan18/DeepLog