Data_Processing

May 7, 2021

1 Sequntial Event Feature Extraction Notebook

1.0.1 Log Structure

The HDFS log is built the following way:

- The date and time
- An unknown 2-4 digit code
- The priority of the logging event (INFO, WARN, DEBUG, ERROR, etc.)
- The source of the logging event message
- Log message

1.1 Import Libraries

[1]:
```
# Parsing and wrangling
import pandas as pd
import numpy as np
import regex as re
```

[3]:
```
## Import Raw Log Data
raw = pd.read_csv('Data/HDFS.log', header=None, sep='\n')[0]
raw.head()
```

[3]:
```
0    081109 203518 143 INFO dfs.DataNode$DataXceive…
1    081109 203518 35 INFO dfs.FSNamesystem: BLOCK*…
2    081109 203519 143 INFO dfs.DataNode$DataXceive…
3    081109 203519 145 INFO dfs.DataNode$DataXceive…
4    081109 203519 145 INFO dfs.DataNode$PacketResp…
Name: 0, dtype: object
```

1.2 Code Parsing Section

Here we'll parse the log file for block IDs which we will use this to aggregate message sequences for the corresponding block ID.

Note the checkpoint cells. Due to memory limitations, checkpoints were created where the kernel could be restarted to free RAM for the next processing cells.

1.3 BLock IDs Anomaly Labels

[9]:
```
## Import Anomaly Labels
labels = pd.read_csv('Data/anomaly_label.csv')
labels.iloc[:,1][labels.Label == 'Anomaly'] = 1
labels.iloc[:,1][labels.Label == 'Normal'] = 0

length = len(labels)
anomalies = len(labels[labels.Label==1])
print('Len labels: ', length)
print('Anomaly count: ', anomalies)
print('% Anomalous: ', round(anomalies/length*100, 2),'%')
```

```
Len labels:  575061
Anomaly count:  16838
% Anomalous:  2.93 %
```

1.4 Extract BlockIDs from Events (rows)

[10]:
```
## Extract Blocks from Log
blocks_in_order = raw.str.findall(r'(blk_.[\d]*)')
unlist_vectorized = np.vectorize(lambda x: x[0])
blocks_in_order = pd.Series(unlist_vectorized(blocks_in_order))


## Label the extracted block ids via conversion dictionary
binarizer = dict(zip(labels.BlockId, labels.Label))
binarizer_vectorized = np.vectorize(lambda x: binarizer[x])
blocks_binarized = pd.Series(binarizer_vectorized(blocks_in_order))

## Create export checkpoint
labeled_blks = pd.DataFrame({'blkID':blocks_in_order, 'anomaly':
 ↪blocks_binarized})
labeled_blks.head()
labeled_blks.to_feather('Data/labeled_blks.feather')
```

[20]:
```
print('Unique blocks in Log File: ', len(blocks_in_order.unique()))
print('Anomalies in the Log File: ', binarizer_vectorized(blocks_in_order.
 ↪unique()).sum()) # anomalous **blocks** in the log file
```

```
Unique blocks in Log File:  575061
Anomalies in the Log File:  16838
```

The number of unique block IDs and anomaly counts in both the parsed log file and labeled file are equal from the printed outputs. This confirms that the parsing was done correctly.

1.5 Extract Raw Log Messages

[5]:
```
# Extract Raw Messages
full_msg = raw.str.extract(r'((?<=:\s).*)')[0]
full_msg.to_csv('Data/full_msg.csv', index=None, header=None)

# CHECKPOINT
full_msg.head()
```

[5]:
```
0    Receiving block blk_-1608999687919862906 src: …
1    BLOCK* NameSystem.allocateBlock: /mnt/hadoop/m…
2    Receiving block blk_-1608999687919862906 src: …
3    Receiving block blk_-1608999687919862906 src: …
4    PacketResponder 1 for block blk_-1608999687919…
Name: 0, dtype: object
```

1.6 Convert Messages Categorical Values

We remove all unique event identifiers to get general message structures, convert them to numeric, and label each log event this way.

[9]:
```
# Import Raw message checkpoint
full_msg = pd.read_csv('Data/full_msg.csv', index_col=False, header=None,␣
↪squeeze=True)
```

[16]:
```
# Split into chunks for easier processing
chunks = 200
chunked_msgs = np.array_split(full_msg, chunks)

# Function to join key words
toSentence = np.vectorize(lambda x: " ".join(x))

str_msgs = pd.Series(dtype='object')
for chunk in range(chunks):
    # Extract and join key words without IDs
    sentences = pd.Series(toSentence(chunked_msgs[chunk].str.
↪findall(r'([A-Za-z]+)')))
    str_msgs = pd.concat([str_msgs, sentences])
    # Save that juicy RAM
    del sentences

print('Messages extracted: ', len(str_msgs))
```

```
Messages extracted:  11175629
```

[23]:
```
coded_msgs = pd.Categorical(str_msgs).codes

print('Unique Message Types: ', len(pd.Series(coded_msgs).unique()))
```

```
coded_msgs
```

```
Unique Message Types:  75
```

[23]:
```
array([55,  4, 55, …, 62, 62, 62], dtype=int8)
```

1.7 Group Message Events by Block ID and Label

[91]:
```python
# Import anomaly labels
labeled_blks = pd.read_feather('Data/labeled_blks.feather')

# Log dataframe of blocks and event category in order
blk_events = pd.DataFrame({'blk_ID': labeled_blks.blkID, 'msg_code':coded_msgs})

# Groupby by block to create event sequences
blk_event_sequences = blk_events.groupby('blk_ID')['msg_code'].apply(list).
 ↪reset_index(name='sequence')

# Assign anomaly labels using a dictionary
blk_key = dict(labels.values)
vectorized_anom_labeler = np.vectorize(lambda x: blk_key[x])

blk_event_sequences['anomaly'] = pd.
 ↪Series(vectorized_anom_labeler(blk_event_sequences.blk_ID))

# Export the golden feature
blk_event_sequences.to_csv('Data/blk_event_sequences.csv', index = False,␣
 ↪header = True)

blk_event_sequences.head()
```

[91]:
```
                      blk_ID  \
0  blk_-1000002529962039464
1   blk_-100000266894974466
2  blk_-1000007292892887521
3  blk_-1000014584150379967
4  blk_-1000028658773048709

                                            sequence  anomaly
0  [55, 55, 55, 16, 51, 53, 51, 53, 3, 3, 3, 51, 53]        0
1  [20, 55, 55, 55, 3, 3, 3, 51, 53, 51, 53, 51, …       0
2  [55, 55, 16, 55, 51, 53, 51, 53, 51, 53, 3, 3, 3]        0
3  [55, 20, 55, 55, 3, 3, 3, 51, 53, 51, 53, 51, …       0
4  [55, 55, 55, 20, 51, 53, 51, 53, 51, 53, 3, 3,…       0
```

[97]:
```python
# Quick Check
labels = pd.read_csv('Data/anomaly_label.csv')
labels.iloc[:,1][labels.Label == 'Anomaly'] = 1
labels.iloc[:,1][labels.Label == 'Normal'] = 0

target_rows = len(labels)
final_rows = len(blk_event_sequences)

original_anomalies = len(labels[labels.Label==1])
final_anomalies = len(blk_event_sequences[blk_event_sequences.anomaly==1])

print(f'Total blocks present {final_rows} of {target_rows}')
print(f'Total anomalies present {final_anomalies} of {original_anomalies}')
```

```
Total blocks present 575061 of 575061
Total anomalies present 16838 of 16838
```