Reducing effort in logs reviews

Udacity Machine Learning Engineer Capstone Project

EDWIN, ANG PAU HUANG

CONTENT

1. Project Definition

1.1. Problem Statement

Logs play an important role in the maintenance of systems. Every day a vast number of logs are generated by the live systems. When a problem occurs, engineers need to examine recorded logs to gain insights into the problem. There is a large number of recurrent issues reflected by the logs, leading to a lot of redundant effort in examining logs and diagnosing the previously known problems. It is very time consuming for engineers to review large quantity of logs through manual examination of the logs.

The amount of time that engineers spent on reviewing logs can be reduced by directing their attention to only logs that are abnormal. Assuming past ground truth data is available on which logs are normal and abnormal, this project explores the use of supervised and unsupervised approaches to detecting abnormal logs

1.2. Project Overview

I'm using AWS Sagemaker to implement machine learning model that detects abnormal log sequences from logs. This project assumes that ground truth data/labels are available on which logs are normal and abnormal. Publicly available dataset is used as follows:

- Logs: HDFS_100k.log_structured.csv[1]
- Labels: abnormal_labels.csv[2]

One supervised machine learning approach and two unsupervised learning approaches will be explored in this project. The general workflow for both supervised and unsupervised approach in this project will be same as follows. The algorithms used is described in section 2.3.

[1] Github: HDFS_100k.log_structured.csv

[2] Github: anomaly_label.csv

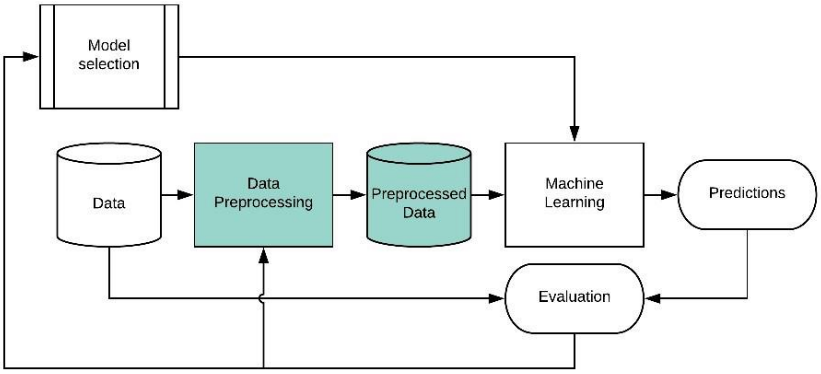


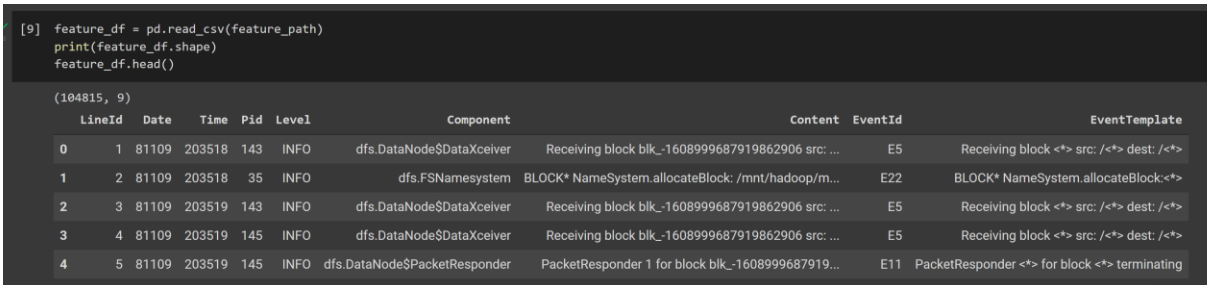
Figure 1: Overall workflow

1.3. Metrics

It is assumed in this project there that by reducing number of false positives, the quantity of logs that engineers need to analyze is reduced. The main evaluation metric to be used will be precision. Recall metric will also be used.

2. Analysis

2.1. Data exploration



```
[9] feature_df = pd.read_csv(feature_path)
    print(feature_df.shape)
    feature_df.head()
```

```
(104815, 9)
```

	LineId	Date	Time	Pid	Level	Component	Content	EventId	EventTemplate
0	1	81109	203518	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	E5	Receiving block <*> src: /<*> dest: /<*>
1	2	81109	203518	35	INFO	dfs.FSNamesystem	BLOCK* NameSystem.allocateBlock: /mnt/hadoop/m...	E22	BLOCK* NameSystem.allocateBlock:<*>
2	3	81109	203519	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	E5	Receiving block <*> src: /<*> dest: /<*>
3	4	81109	203519	145	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: ...	E5	Receiving block <*> src: /<*> dest: /<*>
4	5	81109	203519	145	INFO	dfs.DataNode$PacketResponder	PacketResponder 1 for block blk_-1608999687919...	E11	PacketResponder <*> for block <*> terminating

Figure 2: HDFS logs

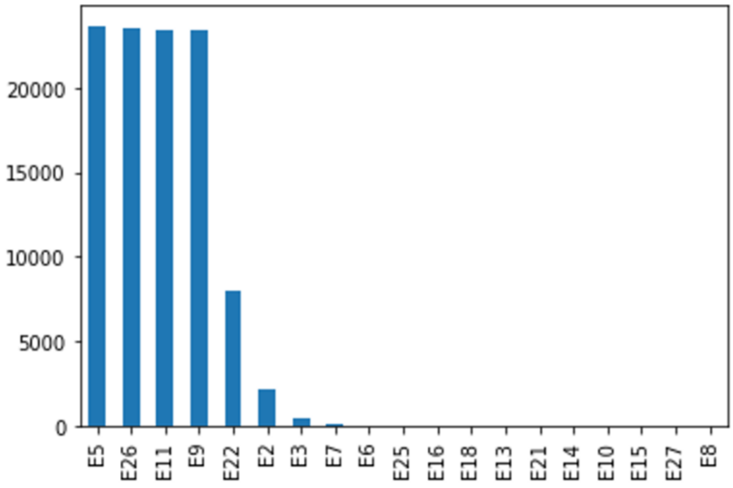
Hadoop File System Logs (HDFS) are used for analysis in this project. There are 184,815 rows of data in HDFS logs, and 9 columns. There are no missing values in the data.

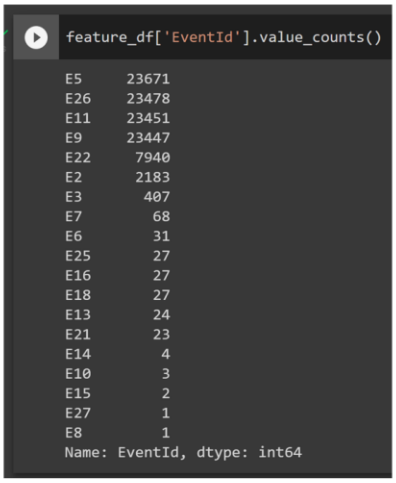
2.2. Exploratory Visualization

The main features that will be selected for use in this project is 'Column' and 'EventId', because this will be used to extract and form log sequences. Section 3.1 elaborates more on how the log sequences are formed.

- Within the feature 'Column' there are 7940 unique/different types of 'blk_id's. 'blk_id' represents TaskID in HDFS logs. It can be considered as a kind of session 'ID'.

- Within the ‘EventId’ columns, there are 19 unique ‘EventId’ types, which is an abstract representation of a log event category. The following diagrams shows the counts per unique event type.





```
feature_df['EventId'].value_counts()

E5      23671
E26     23478
E11     23451
E9      23447
E22      7940
E2       2183
E3        407
E7         68
E6         31
E25        27
E16        27
E18        27
E13        24
E21        23
E14         4
E10         3
E15         2
E27         1
E8          1
Name: EventId, dtype: int64
```

Figure 3: Unique events types

2.3. Algorithm and Techniques

Three modelling approaches were selected in this project:

- Supervised approach to aid in abnormal logs detection
  1. Linear Learner - Given that ground truth data on abnormal events is available, Linear Learner AWS Sagemaker built-in algorithms was used to detect abnormal events.
- Unsupervised approach to aid in abnormal logs detection
  2. LogCluster - in AWS Sagemaker platform, I used a custom library called LogCluster from loglizer[3] to explore unsupervised approach of clustering to aid in detection of abnormal logs. This clustering approach from Microsoft research is based on agglomerative hierarchical clustering,
  3. HDBSCAN - Separately, I explored the use of another unsupervised approach of clustering using Hierarchical Density-based Spatial Clustering of Applications with Noise[4] (HDBSCAN). It extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters.

[3] Logizer github

[4] How HDBSCAN Works

2.4. Benchmark

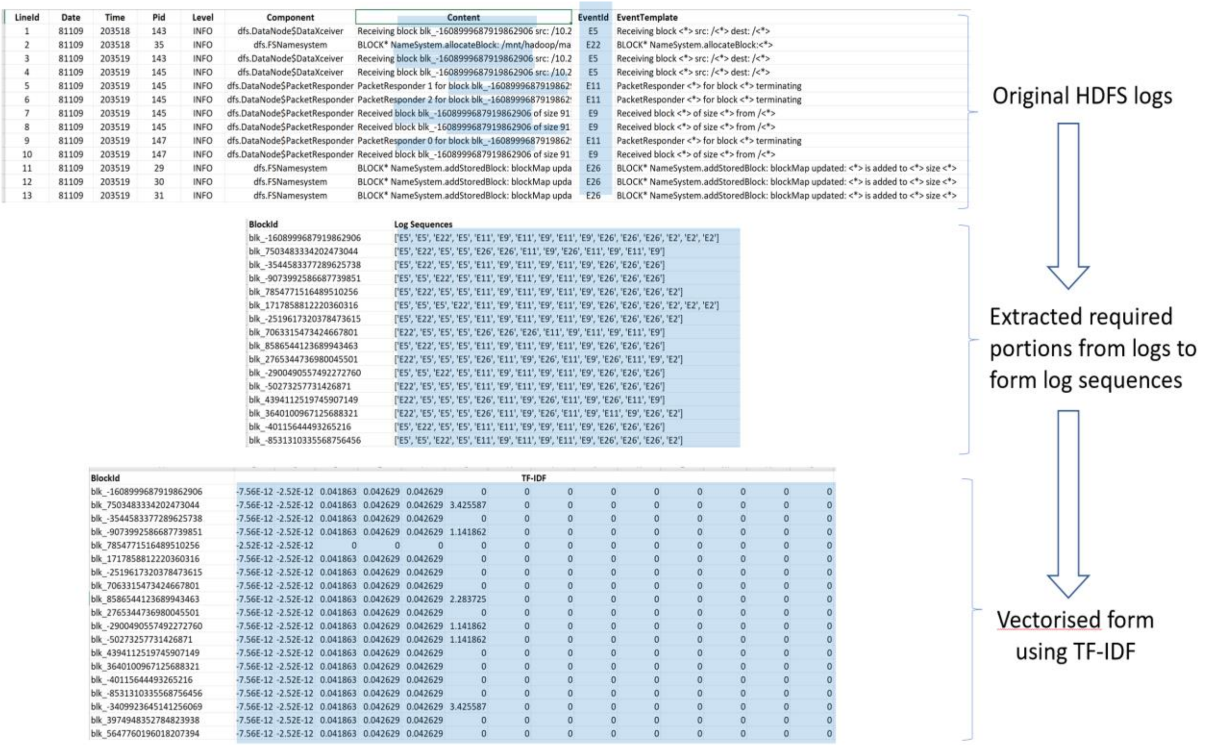
There isn't a benchmark model available for the dataset used in this project. The three approaches to be implemented (supervised, supervised) will be compared against each other using the evaluation metrics.

In production scenario, it is unlikely that ground truth data on abnormal logs labels is available. The more possible approach is the unsupervised learning approach. Hence the unsupervised learning approach of LogCluster which was based on Microsoft research will serve as the reference for the other approaches.

3. Methodology

3.1. Data Preprocessing

The use of log sequences helps to achieve higher precision in abnormalities detection according to Shang el[5]. The following diagram illustrates how data is preprocessed for models' inputs by forming log sequences.



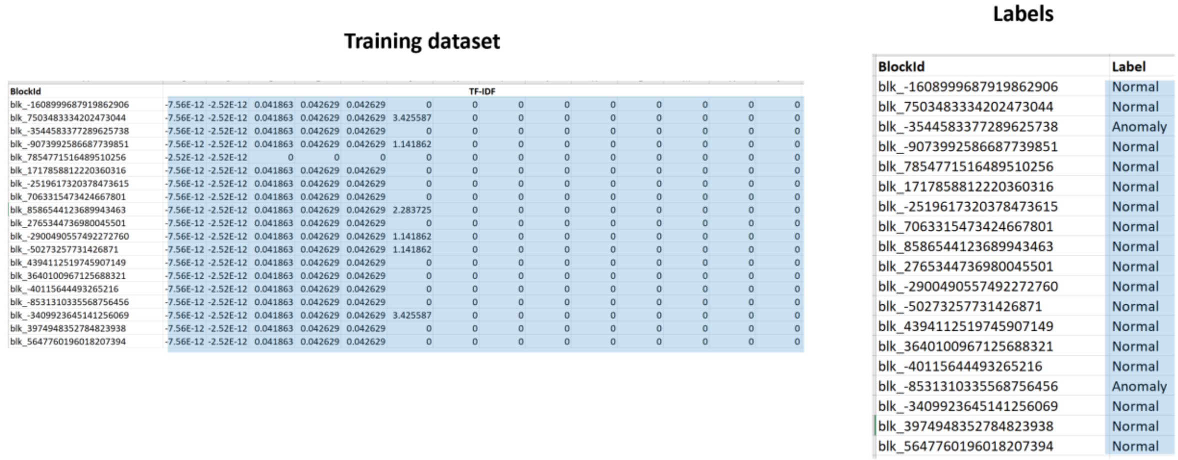
LineId	Date	Time	Pid	Level	Component	Content	EventId	EventTemplate
1	81109	203518	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: /10.2	E5	Receiving block <*> src: /<*> dest: /<*>
2	81109	203518	35	INFO	dfs.FSNamesystem	BLOCK* NameSystem.allocateBlock: /mnt/hadoop/ma	E22	BLOCK* NameSystem.allocateBlock:<*>
3	81109	203519	143	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: /10.2	E5	Receiving block <*> src: /<*> dest: /<*>
4	81109	203519	145	INFO	dfs.DataNode$DataXceiver	Receiving block blk_-1608999687919862906 src: /10.2	E5	Receiving block <*> src: /<*> dest: /<*>
5	81109	203519	145	INFO	dfs.DataNode$PacketResponder	PacketResponder 1 for block blk_-160899968791986	E11	PacketResponder <*> for block <*> terminating
6	81109	203519	145	INFO	dfs.DataNode$PacketResponder	PacketResponder 2 for block blk_-160899968791986	E11	PacketResponder <*> for block <*> terminating
7	81109	203519	145	INFO	dfs.DataNode$PacketResponder	Received block blk_-1608999687919862906 of size 91	E9	Received block <*> of size <*> from /<*>
8	81109	203519	145	INFO	dfs.DataNode$PacketResponder	Received block blk_-1608999687919862906 of size 91	E9	Received block <*> of size <*> from /<*>
9	81109	203519	147	INFO	dfs.DataNode$PacketResponder	PacketResponder 0 for block blk_-160899968791986	E11	PacketResponder <*> for block <*> terminating
10	81109	203519	147	INFO	dfs.DataNode$PacketResponder	Received block blk_-1608999687919862906 of size 91	E9	Received block <*> of size <*> from /<*>
11	81109	203519	29	INFO	dfs.FSNamesystem	BLOCK* NameSystem.addStoredBlock: blockMap upda	E26	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
12	81109	203519	30	INFO	dfs.FSNamesystem	BLOCK* NameSystem.addStoredBlock: blockMap upda	E26	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
13	81109	203519	31	INFO	dfs.FSNamesystem	BLOCK* NameSystem.addStoredBlock: blockMap upda	E26	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>

BlockId	Log Sequences
blk_-1608999687919862906	['E5', 'E5', 'E22', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26', 'E2', 'E2', 'E2']
blk_7503483334202473044	['E5', 'E22', 'E5', 'E5', 'E26', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9', 'E11', 'E9']
blk_-3544583377289625738	['E5', 'E22', 'E5', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_-9073992586687739851	['E5', 'E5', 'E22', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_7854771516489510256	['E5', 'E22', 'E5', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26', 'E2']
blk_1717858812220360316	['E5', 'E5', 'E5', 'E22', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26', 'E2', 'E2', 'E2']
blk_-2519617320378473615	['E5', 'E22', 'E5', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26', 'E2']
blk_7063315473424667801	['E22', 'E5', 'E5', 'E5', 'E26', 'E26', 'E26', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9']
blk_8586544123689943463	['E5', 'E22', 'E5', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_2765344736980045501	['E22', 'E5', 'E5', 'E5', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9', 'E2']
blk_-2900490557492272760	['E5', 'E5', 'E22', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_-50273257731426871	['E22', 'E5', 'E5', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_4394112519745907149	['E22', 'E5', 'E5', 'E5', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9']
blk_3640100967125688321	['E22', 'E5', 'E5', 'E5', 'E26', 'E11', 'E9', 'E26', 'E11', 'E9', 'E11', 'E9', 'E26', 'E2']
blk_-40115644493265216	['E5', 'E22', 'E5', 'E5', 'E11', 'E11', 'E9', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26']
blk_-8531310335568756456	['E5', 'E5', 'E22', 'E5', 'E11', 'E9', 'E11', 'E9', 'E11', 'E9', 'E26', 'E26', 'E26', 'E2']

BlockId	TF-IDF													
blk_-1608999687919862906	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7503483334202473044	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_-3544583377289625738	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-9073992586687739851	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_7854771516489510256	-2.52E-12	-2.52E-12	0	0	0	0	0	0	0	0	0	0	0	0
blk_1717858812220360316	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2519617320378473615	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7063315473424667801	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_8586544123689943463	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	2.283725	0	0	0	0	0	0	0	0
blk_2765344736980045501	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2900490557492272760	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_-50273257731426871	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_4394112519745907149	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_3640100967125688321	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-40115644493265216	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-8531310335568756456	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-3409923645141256069	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_3974948352784823938	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_5647760196018207394	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0

Figure 4: How HDFS logs are processed

[5] Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds



Training dataset

BlockId														
							TF-IDF							
blk_-1608999687919862906	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7503483334202473044	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_-3544583377289625738	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-9073992586687739851	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_7854771516489510256	-2.52E-12	-2.52E-12	0	0	0	0	0	0	0	0	0	0	0	0
blk_1717858812220360316	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2519617320378473615	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7063315473424667801	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_8586544123689943463	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	2.283725	0	0	0	0	0	0	0	0
blk_2765344736980045501	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2900490557492272760	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_-50273257731426871	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_4394112519745907149	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_3640100967125688321	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-40115644493265216	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-8531310335568756456	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-3409923645141256069	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_3974948352784823938	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_5647760196018207394	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0

Labels

BlockId	Label
blk_-1608999687919862906	Normal
blk_7503483334202473044	Normal
blk_-3544583377289625738	Anomaly
blk_-9073992586687739851	Normal
blk_7854771516489510256	Normal
blk_1717858812220360316	Normal
blk_-2519617320378473615	Normal
blk_7063315473424667801	Normal
blk_8586544123689943463	Normal
blk_2765344736980045501	Normal
blk_-2900490557492272760	Normal
blk_-50273257731426871	Normal
blk_4394112519745907149	Normal
blk_3640100967125688321	Normal
blk_-40115644493265216	Normal
blk_-8531310335568756456	Anomaly
blk_-3409923645141256069	Normal
blk_3974948352784823938	Normal
blk_5647760196018207394	Normal

Figure 5: Training dataset and labels

From HDFS logs, task ids represented by 'blk_id' are extracted using regular expression from the column 'Content'. Events with the same task ID are linked together to form log sequences. Events are represented under column 'EventId'. The extracted log sequences are then vectorized using TF-IDF[6].

70% of the processed data was set aside for training the models, while 30% was used for testing in this project.

3.2. Implementation and Refinement

The syllabus of Udacity's Machine Learning Nanodegree programme course was on how to use AWS Sagemaker in implementing machine learning techniques. The following describes on ML techniques implementation using AWS Sagemaker in this project.

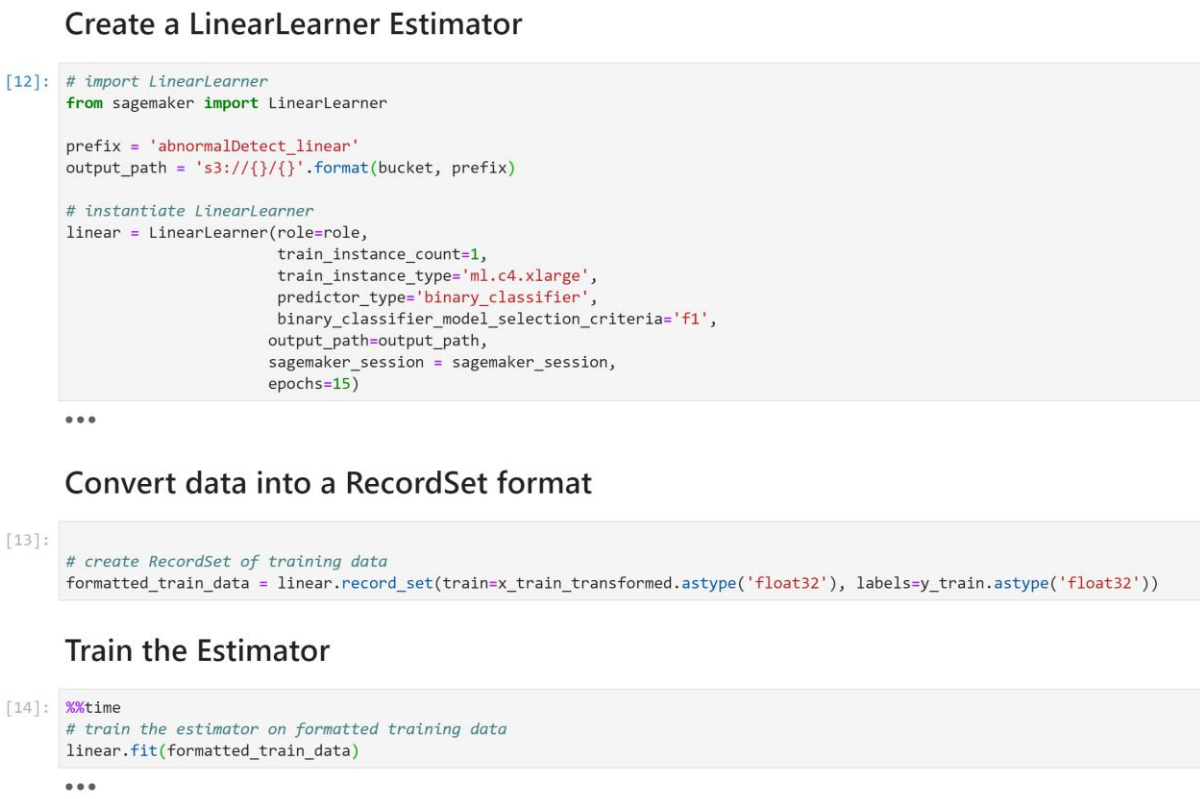
3.2.1 Supervised Learning Approach – Linear Learner

Amazon SageMaker Python SDK provides framework estimators to train ML models. A LinearLearner[7] estimator was instantiated using the SDK.

The training data (ie. vectorized log sequences) was converted into Recordset object instances, stored in Amazon S3, which is a data input format accepted by AWS Sagemaker Estimators. The LinearLearner model was then trained using the formatted training data.

[6] tf–idf - Wikipedia

[7] AWS Sagemaker Linear Learner



Create a LinearLearner Estimator

[12]:

```
# import LinearLearner
from sagemaker import LinearLearner

prefix = 'abnormalDetect_linear'
output_path = 's3://{}/{}'.format(bucket, prefix)

# instantiate LinearLearner
linear = LinearLearner(role=role,
                       train_instance_count=1,
                       train_instance_type='ml.c4.xlarge',
                       predictor_type='binary_classifier',
                       binary_classifier_model_selection_criteria='f1',
                      output_path=output_path,
                      sagemaker_session = sagemaker_session,
                      epochs=15)
```

•••

Convert data into a RecordSet format

[13]:

```
# create RecordSet of training data
formatted_train_data = linear.record_set(train=x_train_transformed.astype('float32'), labels=y_train.astype('float32'))
```

Train the Estimator

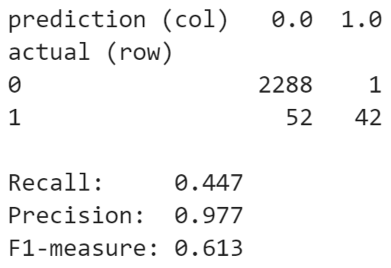
[14]:

```
%%time
# train the estimator on formatted training data
linear.fit(formatted_train_data)
```

•••

Figure 6: Linear Learner model setup and training

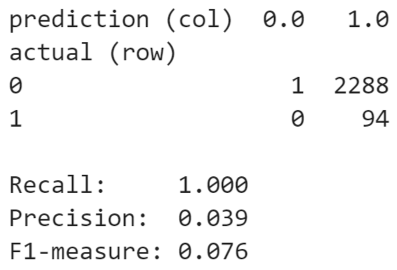
After testing the trained model with test data, the initial results are as follows:



```
prediction (col)   0.0  1.0
actual (row)
0                 2288    1
1                   52   42

Recall:     0.447
Precision:  0.977
F1-measure: 0.613
```

Figure 7: Linear Learner model results

To work with imbalanced dataset, LinearLearner has a hyperparameter called `positive_example_weight_mult` which supposedly can be used to improve performance of model of an imbalanced dataset. It adjusts the weight so that errors in classifying negative vs. positive examples have equal impact on training loss. For refinement, this feature was explored and used in setup of another estimator. The separate model was trained and tested. The result however was a poorer performance in model as follows in terms of precision and F1 score.



```
prediction (col)  0.0   1.0
actual (row)
0                   1  2288
1                   0    94

Recall:     1.000
Precision:  0.039
F1-measure: 0.076
```

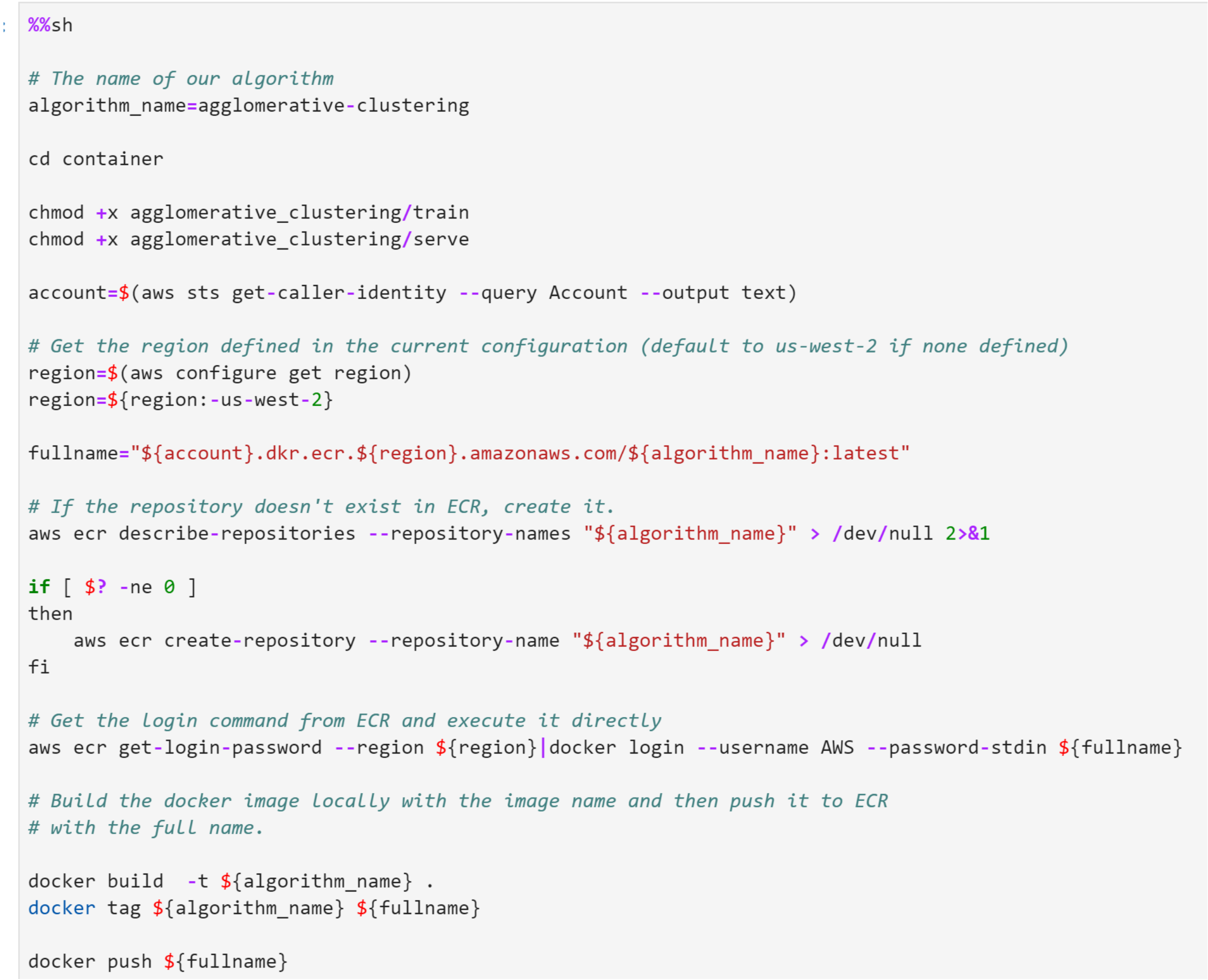
Figure 8: Linear Learner model results after fine tuning attempt

3.2.2 Unsupervised Learning Approaches

There are two unsupervised learning approaches implemented in this section. There is a need for use of containers in implementation of non builit-in ML algorithms in AWS Sagemaker. Much effort on my part was spent in learning about Docker, how to build containers, and then about how to use different AWS components required to deploy non AWS Sagemaker built-in algorithms in this project.

The following shell code builds the container image using `docker build` and push the container image to Amazon Elastic Container Registry (ECR) using `docker push`. This code looks for an ECR repository. If the repository doesn't exist, the script will create it. The code is largely similar for both unsupervised approaches implemented in this section.

Building and registering the container



```
%%sh

# The name of our algorithm
algorithm_name=agglomerative-clustering

cd container

chmod +x agglomerative_clustering/train
chmod +x agglomerative_clustering/serve

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none defined)
region=$(aws configure get region)
region=${region:-us-west-2}

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.
aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1

if [ $? -ne 0 ]
then
    aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi

# Get the login command from ECR and execute it directly
aws ecr get-login-password --region ${region}|docker login --username AWS --password-stdin ${fullname}

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

docker build  -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}
```

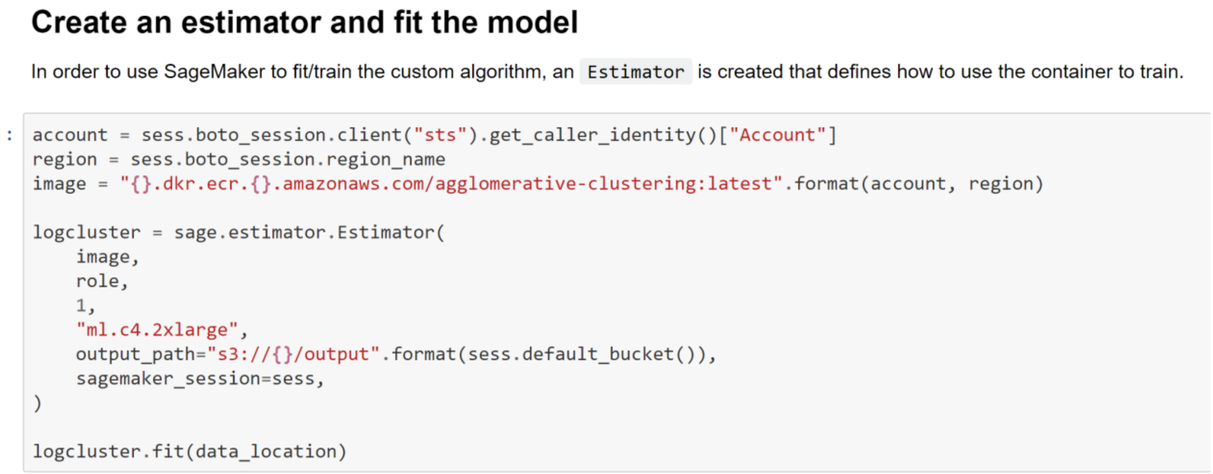
Figure 9: Container build and push to Amazon ECR

3.2.2.1 Log Cluster

LogCluster[8] algorithm was developed by a Microsoft Research team. It is largely based on agglomerative hierarchical clustering techinique. An estimator using LogCluster

[8] Log Clustering based Problem Identification for Online Service Systems

algorithm was instantiated using the AWS SDK.



Create an estimator and fit the model

In order to use SageMaker to fit/train the custom algorithm, an `Estimator` is created that defines how to use the container to train.

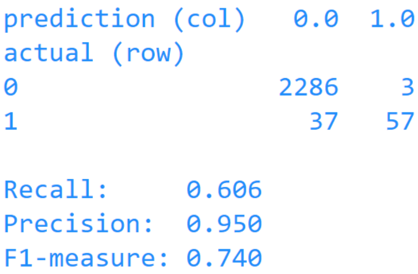
```
account = sess.boto_session.client("sts").get_caller_identity()["Account"]
region = sess.boto_session.region_name
image = "{}.dkr.ecr.{}.amazonaws.com/agglomerative-clustering:latest".format(account, region)

logcluster = sage.estimator.Estimator(
    image,
    role,
    1,
    "ml.c4.2xlarge",
    output_path="s3://{}/output".format(sess.default_bucket()),
    sagemaker_session=sess,
)

logcluster.fit(data_location)
```

Figure 10: Log Cluster model setup and training

Attempt was made to tune the model by training the model with different combinations of hyperparameters ‘max_dist’ and ‘anomaly_threshold’.

- The ‘max_dist’ hyperparameter influences the number of clusters being formed. The ‘anomaly_threshold’ hyperparameter is used by the model to predict whether a data point is an abnormality, by comparing the distance from known clusters and the ‘anomaly_threshold’ set by this hyperparameter.
- Discrete values of [0.2 0.3, 0.4] were tested for both ‘max_dist’ and ‘anomaly_threshold’. The accuracy results for the different combinations of hyperparameters were the same.
- Hyperparameters values of 0.3 was then selected for both ‘max_dist’ and ‘anomaly_threshold’ during model training.

After the model was trained and tested with test data, the following illustrates the results.

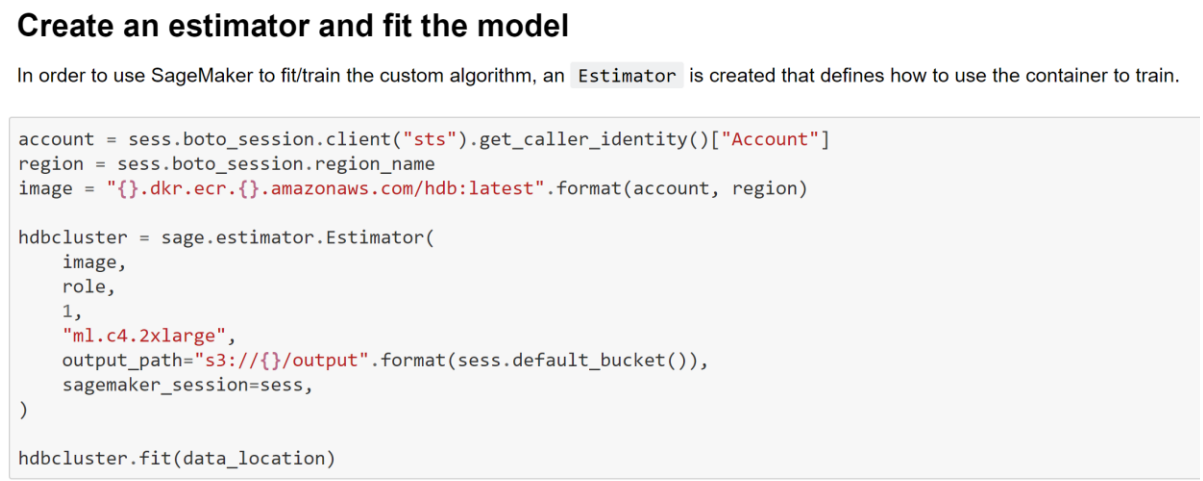


```
prediction (col)   0.0  1.0
actual (row)
0                 2286    3
1                   37   57

Recall:     0.606
Precision:  0.950
F1-measure: 0.740
```

Figure 11: Log Cluster model results

3.2.2.2 HDBSCAN

HDBSCAN is a clustering algorithm developed by Campello, Moulavi, and Sander[9]. It extends Density-Based Spatial Clustering of Applications with Noise (DBSCAN) by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters . An estimator using HDBSCAN algorithm was instantiated using the AWS SDK.



Create an estimator and fit the model

In order to use SageMaker to fit/train the custom algorithm, an `Estimator` is created that defines how to use the container to train.

```
account = sess.boto_session.client("sts").get_caller_identity()["Account"]
region = sess.boto_session.region_name
image = "{}.dkr.ecr.{}.amazonaws.com/hdb:latest".format(account, region)

hdbcluster = sage.estimator.Estimator(
    image,
    role,
    1,
    "ml.c4.2xlarge",
    output_path="s3://{}/output".format(sess.default_bucket()),
    sagemaker_session=sess,
)

hdbcluster.fit(data_location)
```

Figure 12: HDBSCAN model setup and training

Using SKlearn GridsearchCV[10], the following range of hyperparameters was tested. Hyperparameters that lead to best performing model is highlighted in gray and selected for use to set up the model for training.

- ‘min_samples’ hyperparameter influences how how conservative we want the clustering to be. The larger the value, the more conservative it is.
- ‘min_cluster_size’ hyperparameter determines the smallest size grouping that we wish to consider a cluster.
- ‘cluster_selection_metrics’ hyperparameter determines how the hdbscan model selects flat clusters from the cluster tree hierarchy
- ‘metrics’ hyperparameter determines the distance metric used by hdbscan model

min_samples	10, 20, 30, 50
min_cluster_size	40, 50, 100
cluster_selection_metrics	eom, leaf
metrics	euclidean, manhattan

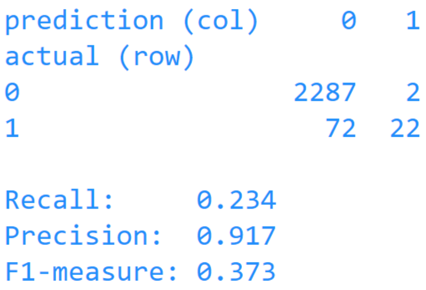
Figure 13: Hyperparameters range used in GridSearchCV

[9] Density-Based Clustering Based on Hierarchical Density Estimates

[10] SKlearn GridsearchCV

Hyperparameters that lead to best performing model is highlighted in gray, and selected for use to set up the model for training.

The hdbscan library supports the GLOSH outlier detection algorithm[11], and does so within the HDBSCAN clustering class. The outlier detection function from hdbscan library was used to predict whether a data point is an abnormality. After the model was trained and tested with test data, the following illustrates the results.



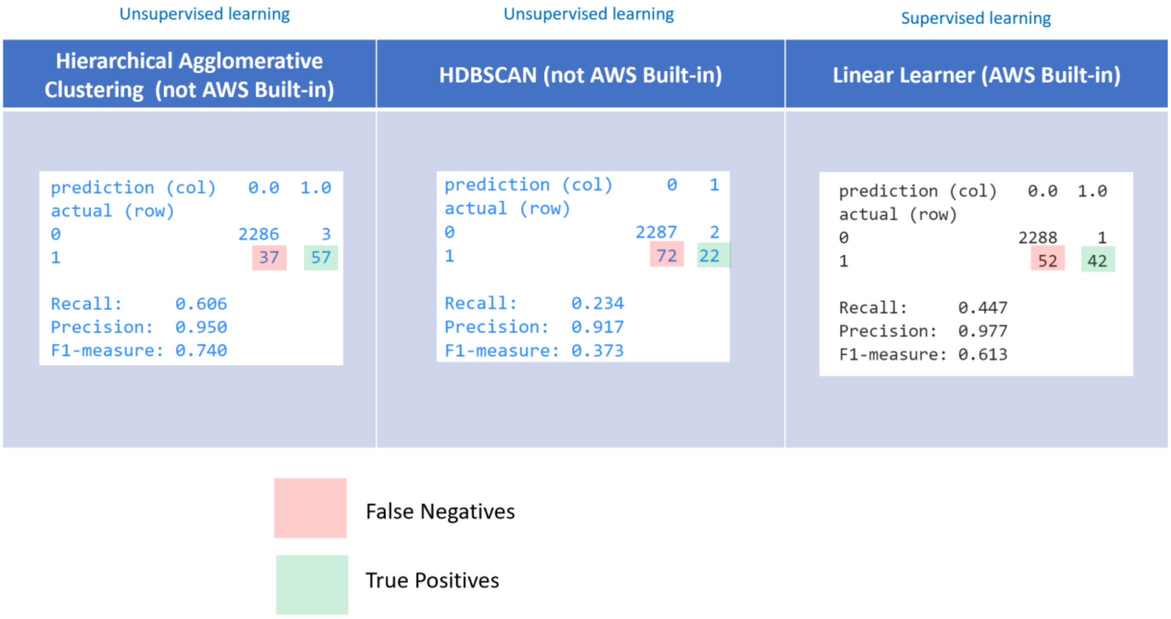
```
prediction (col)     0    1
actual (row)
0                 2287    2
1                   72   22

Recall:     0.234
Precision:  0.917
F1-measure: 0.373
```

Figure 14: HDBSCAN model results

4. Results

4.1. Model evaluation and Validation

The following diagram allows a comparison of results between the different models implemented.



Unsupervised learning
Unsupervised learning
Supervised learning
Hierarchical Agglomerative Clustering (not AWS Built-in)
HDBSCAN (not AWS Built-in)
Linear Learner (AWS Built-in)

```
prediction (col)   0.0  1.0
actual (row)
0                 2286    3
1                   37   57

Recall:     0.606
Precision:  0.950
F1-measure: 0.740
```

```
prediction (col)     0    1
actual (row)
0                 2287    2
1                   72   22

Recall:     0.234
Precision:  0.917
F1-measure: 0.373
```

```
prediction (col)   0.0  1.0
actual (row)
0                 2288    1
1                   52   42

Recall:     0.447
Precision:  0.977
F1-measure: 0.613
```

🟥 False Negatives

🟩 True Positives

Figure 15: Model performance comparison

[11] HDBSCAN outlier detection

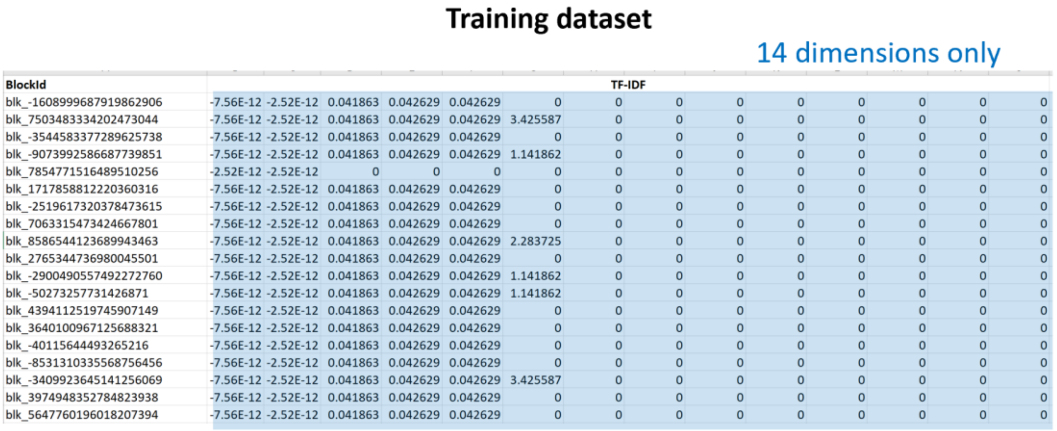
Using precision and F1-measure as the metric for model comparison, hierarchical agglomerative clustering is the best performing approach based on the HDFS logs used in this project. This model approach also have the least number of false negatives.

4.2. Justification

In general, there are no labels (normal or abnormal) available for logs in systems in enterprise systems available. Hence unsupervised approach is a more feasible approach to explore.

There are different types of logs that exists in enterprise systems. For the use case of HDFS logs, it is shown that hierarchical agglomerative clustering is the best approach for this project's HDFS dataset, and generally a good approach to explore for use in production environment. There are other types of logs like Windows OS, Linux OS logs that exists. Hierarchical agglomerative clustering may not necessarily be a good approach to explore for use in production.

The dimensions of training dataset based on HDFS logs used in this project is around 14.



Training dataset

14 dimensions only

BlockId							TF-IDF							
blk_-1608999687919862906	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7503483334202473044	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_-3544583377289625738	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-9073992586687739851	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_7854771516489510256	-2.52E-12	-2.52E-12	0	0	0	0	0	0	0	0	0	0	0	0
blk_1717858812220360316	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2519617320378473615	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_7063315473424667801	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_8586544123689943463	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	2.283725	0	0	0	0	0	0	0	0
blk_2765344736980045501	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-2900490557492272760	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_-50273257731426871	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	1.141862	0	0	0	0	0	0	0	0
blk_4394112519745907149	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_3640100967125688321	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-40115644493265216	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-8531310335568756456	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_-3409923645141256069	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	3.425587	0	0	0	0	0	0	0	0
blk_3974948352784823938	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0
blk_5647760196018207394	-7.56E-12	-2.52E-12	0.041863	0.042629	0.042629	0	0	0	0	0	0	0	0	0

Figure 16: Number of dimensions used for HDFS based training dataset

The way log review reports are 'vectorized' will be very different in other system logs. TF-IDF may not the technique used. For Windows and Linux OS logs, the dimensions of the dataset can be very high (in terms of a few hundreds or more).

For low dimension dataset like HDFS, it is inferred from results of this project that hierarchical agglomerative clustering is a good approach to explore further for use.

For high dimension datasets like Window OS or Linux OS, HDBSCAN can be explored further for the following reasons:

1. The number of dimensions for HFDS logs is only 14. For log review reports, the **number of dimensions is expected to be very high** (~500) and will differs for each log review report type.
2. Hierarchical clustering approaches seem to work better (vs flat *like Kmeans, DBSCAN*) with **high dimensions**.
3. HDBSCAN is **robust with dataset noise**.
   a. Hierarchical agglomerative clustering is sensitive to noise.
4. As log review reports accumulates through time, the quantity of data will eventually be high. HDBSCAN **works better with high quantity of data**
   a. Hierarchical agglomerative clustering may not work well with large quantity of data
5. HDBSCAN **does not have many hyperparameters to tune** -> easier for use in operations

4.3. Conclusion

This project was based on AWS Sagemaker environment.

- I had explored the implementation of both AWS Sagemaker builit-in and non-AWS Sagemaker builit-in algorithms.
- I explored the use of supervised and unsupervised approaches to address the problem reducing effort for engineers in manual log reviews. Unsupervised approaches are in general the more feasible approach in log reviews due to the lack of labels availability.
- The dimensions of eventual dataset (logs) can be low or high. After data processing, the dataset used in this project has low dimension. After tests and model comparisons, hierarchical agglomerative clustering is the best approach. For high dimension datasets (Windows/Linus OS eg not tested in this project), HDBSCAN is a possible approach to test and explore further.