Effective test scrubbing with machine learning and Python

Marius Liaeen, Thomas Hanssen Nornes, Cisco
Carl Martin Rosenberg, Simula Research Laboratory/Certus
Introduction

• Smart scrubbing of logs, what the big deal?

• Continuous integration
  • Lots of testing
  • Fast feedback
Situation

• ~20 000 integration test results every day
• 3-5% failing
Problem

1. how to detect the unknown problems?
   or
2. how to detect the known problems?
Analysing test failures – so far..

• The standard toolbox
  • Logs pattern detection
  • Automatic crash detection

• Result
  • Endless manual inspection of the same problems again and again.
  • Many unscrubbed failing test results
  • Problems are detected too late
Goal: reduce manual inspection

• Avoid inspection of known problems
• Only spend time on unknown problems
Infrastructure failure
A new problem

Infrastructure failure
A new problem

Infrastructure failure
Infrastructure failure

A new problem

Unknown problem

Very important problem

Old recurring problem

Sporadic problem

Very important problem

Infrastructure failure
Unknown problem

Manual inspection
Solution:
Use logs to group similar runs

• Grouping
  • Extracting features from test logs
  • Use Machine Learning to group them

• Assumption
  • Grouped test logs has the same problem

• Savings
  • One known member is enough: Skip inspection of group.
  • Unknown problem: Inspect one.
About test logs

• Follows some «standardized» output
• Different test runs have different runtime “words”
• Sequence might vary
• Differing log files directly does not work!

• Jupyter-time!
<table>
<thead>
<tr>
<th>File</th>
<th>Fetching</th>
<th>Path</th>
<th>Skipped</th>
<th>Main</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>Hangup</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>Memory</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Training set

Trains

Classifier
<table>
<thead>
<tr>
<th>File</th>
<th>Fetching</th>
<th>Path</th>
<th>Skipped</th>
<th>Main</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Hangup</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Network</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Test set**

Classifier

Class

Hangup

Memory
# Training set

<table>
<thead>
<tr>
<th>File</th>
<th>Fetching</th>
<th>Path</th>
<th>Skipped</th>
<th>Main</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

...     ...     ...     ...     ...     ...

- **Trains**
- **Clusterer**
It’s code, after all
#!/usr/bin/env python3

from glob import glob

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

import hdbscan

def create_clusters(filenames):
    return Pipeline([('feature creation', CountVectorizer(input='filename')),
                     ('clustering', hdbscan.HDBSCAN())]).fit_predict(filenames)

create_clusters(glob('dataset/*.dat'))
#!/usr/bin/env python3

from glob import glob

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

import hdbscan

def create_clusters(filenames):
    return Pipeline([('feature creation', CountVectorizer(input='filename')), ('clustering', hdbscan.HDBSCAN())]).fit_predict(filenames)

create_clusters(glob('dataset/*.dat'))
#!/usr/bin/env python3

from glob import glob

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

import hdbscan

def create_clusters(filenames):
    return Pipeline([('feature creation', CountVectorizer(input='filename')),
                     ('clustering', hdbscan.HDBSCAN())]).fit_predict(filenames)

create_clusters(glob('dataset/*.dat'))
#!/usr/bin/env python3

from glob import glob

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

import hdbscan

def create_clusters(filenames):
    return Pipeline([('feature creation', CountVectorizer(input='filename')),
                     ('clustering', hdbscan.HDBSCAN())]).fit_predict(filenames)

create_clusters(glob('dataset/*.dat'))
Convert a collection of text documents to a matrix of token counts

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

Read more in the User Guide.

**Parameters:**

- **input**: string {'filename', 'file', 'content'}
  
  If 'filename', the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.

  If 'file', the sequence items must have a 'read' method (file-like object) that is called to fetch the bytes in memory.

  Otherwise the input is expected to be the sequence strings or bytes items are expected to be analyzed directly.

  **encoding**: string, 'utf-8' by default.

  If bytes or files are given to analyze, this encoding is used to decode.

  **decode_error**: {'strict', 'ignore', 'replace'}
• 7 categorical parameters
• 3 numeric parameters
• 2 parameters accepting *any iterable*
• 1 parameter accepting a numeric range (i.e. \((m, n)\))
• … and 4 (!) parameters accepting *any callable*
#!/usr/bin/env python3

from glob import glob
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
import hdbscan

def create_clusters(filenames):
    return Pipeline([('feature creation', CountVectorizer(input='filename')),
                     ('clustering', hdbscan.HDBSCAN())]).fit_predict(filenames)

create_clusters(glob('dataset/*.dat'))
#!/usr/bin/env python3

from glob import glob

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline

import hdbscan

def create_clusters(filenames):
    return Pipeline(["feature creation", CountVectorizer(input='filename')],
                    ["clustering", hdbscan.HDBSCAN()]).fit_predict(filenames)

create_clusters(glob('dataset/* .dat'))
class hdbSCAN.min_cluster_size=5, min_samples=None, metric='euclidean',
alpha=1.0, p=None, algorithm='best', leaf_size=40, memory=Memory(cachedir=None),
approx_min_span_tree=True, gen_min_span_tree=False, core_dist_n_jobs=4,
cluster_selection_method='eom', allow_single_cluster=False, prediction_data=False,
match_reference_implementation=False, **kwargs)

Perform HDBSCAN clustering from vector array or distance matrix.

HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise. Performs
DBSCAN over varying epsilon values and integrates the result to find a clustering that gives the
best stability over epsilon. This allows HDBSCAN to find clusters of varying densities (unlike
DBSCAN), and be more robust to parameter selection.

min_cluster_size : int, optional (default=5)
The minimum size of clusters; single linkage splits that contain fewer points than this will be
considered points “falling out” of a cluster rather than a cluster splitting into two new
clusters.

min_samples : int, optional (default=None)
The number of samples in a neighbourhood for a point to be considered a core point.

metric : string, or callable, optional (default='euclidean')
The metric to use when calculating distance between instances in a feature array. If metric is
a string or callable, it must be one of the options allowed by
metrics.pairwise.pairwise_distances for its metric parameter. If metric is “precomputed", X is
assumed to be a distance matrix and must be square.

p : int, optional (default=None)
p value to use if using the minkowski metric.

alpha : float, optional (default=1.0)
A distance scaling parameter as used in robust single linkage. See [3] for more information.
Don’t worry too much, though…
scikit-learn Tutorials

An introduction to machine learning with scikit-learn
- Machine learning: the problem setting
- Loading an example dataset
- Learning and predicting
- Model persistence
- Conventions

A tutorial on statistical-learning for scientific data processing
- Statistical learning: the setting and the estimator object in scikit-learn
- Supervised learning: predicting an output variable from high-dimensional observations
- Model selection: choosing estimators and their parameters
- Unsupervised learning: seeking representations of the data
- Putting it all together
- Finding help

Working With Text Data
Measure, discuss, iterate
• Clarify expectations
• Write tests - it’s code, after all
• Crack open the black boxes!
• Measure, discuss, iterate!
“Do machine learning like the great engineer you are, not like the great machine learning expert you aren’t.”

–Martin Zinkevich, Google
Resources

• Coursera.org
  • free ML courses. Especially Andrew Ng’s ML course is good.
• DataCamp.com
  • practical courses on using Python (or R) on different data engineering and ML tasks
• Python3
  • Scikit learn, NumPy, pandas, pyplot
  • Jupyter
  • Anaconda distribution

• This talk and notebook  https://bitbucket.org/ml4af/ndc2017_talk