

Automating Build Failure Analysis Using Semantic Embeddings

Justin Merkel, Luke Daley



What are we automating and why?

Analyzing build and test failures

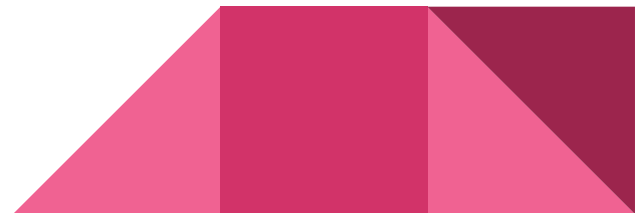
- ❑ Looking for keywords
- ❑ Gathering context
- ❑ Identifying relationships to other failures
- ❑ Looking for path to root cause

Significantly accelerated by expert and historical knowledge.



Why automate failure analytics?

- ❑ **Faster troubleshooting**
 - ❑ More context
 - ❑ Identify the important information
 - ❑ Associate with known/potential fixes
- ❑ **More accurate data at scale**
 - ❑ Reduce toil
 - ❑ Data-driven prioritization
 - ❑ Trending and tracking
 - ❑ Mining and research





Figuring out how to approach the
problem

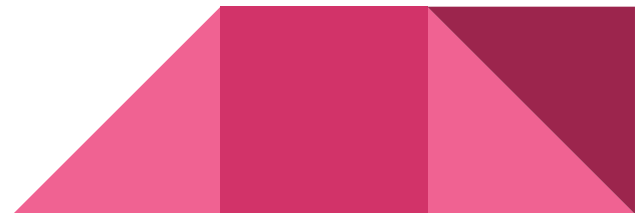
Why base our approach on text data?

- ❑ We started with a more “structured data” approach, but this was quickly determined to not capture enough information about the failure to be useful
- ❑ Text data quickly became our data of choice, because...
 - a. All build failures have some descriptive text in the form of exception/error messages
 - b. It is typically expected that information about the failure is present in the logs
 - c. Text is extremely rich and descriptive



Initial attempts

- ❑ We started to work with simple, NLP-based approaches
- ❑ These methods are often based on word counts within a piece of text
- ❑ These approaches can provide decent results in some cases, but never work as a holistic solution



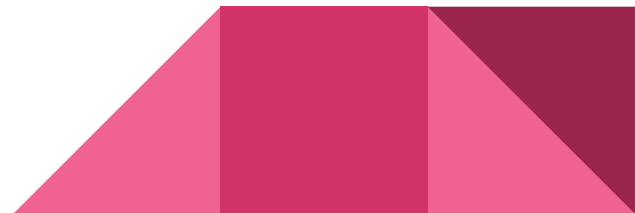
Process 'node' failed with exit code 1	Process 'node' failed with exit code 127
Client count not connect to the server	Server could not connect to the client
Test failed	Test failed unexpectedly
Expected: "Could not connect to database" actual: "null"	Could not connect to database
ReadTimeoutException timeout occurred	JobTimeoutException timeout occurred

What does this mean?

We know that text contains rich information about failures that occurred

BUT

Text data can be messy, complex, and often relies on domain expertise and context





Semantic embeddings for failures

Failure embeddings – what are they?

- ❑ Embeddings are vectors of numbers which capture the meaning of a piece of text.
- ❑ We train an language model to generate embeddings which capture the meaning of a piece of text **in the context of build failure analysis**.
 - We aim to create embedding vectors which represent the **Action and Fault** present in the text chunk.
- ❑ The meaning of an embedding can efficiently be compared to the meaning of another embedding by calculating the cosine similarity.



Failure embeddings – how do they work?

$\text{cosSim}(\text{Invalid HTTP response \{status code: 500\}}, \text{Invalid HTTP response \{status code: 401\}}) = 0.43$

$\text{cosSim}(\text{Invalid HTTP response \{status code: 500\}}, \text{Problem accessing API, message: Internal Server Error}) = 0.87$


$\text{cosSim}(\text{Invalid HTTP response \{status code: 500\}}, \text{Compilation failed, see logs for details}) = 0.01$

Failure embeddings – how are they generated?

- ❑ Embeddings are generated by a bi-encoder language model, trained on a dataset of pairs of text and an associated similarity score.
- ❑ We start with fine-tune a pre-trained model, then fine-tune it to work optimally for failure analytics tasks.
- ❑ Text chunks in the training data are collected from a variety of different projects with a variety of different languages and frameworks.



Failure embeddings – what can we use them for?

- ❑ Effective failure embeddings could be used as the foundation for automating a number of useful failure analytics workflows...
 - a. Categorization of failures into pre-defined groups
 - b. Creating groups of related failures
 - c. User specified failure categories and groups
 - d. Advanced failure search
 - e. Deeper test failure analytics
 - f. Root cause analysis
 - g. And more...
- 



How does this work on real data?

Build: "abcdefg", Failure: #1

Exceptions ->

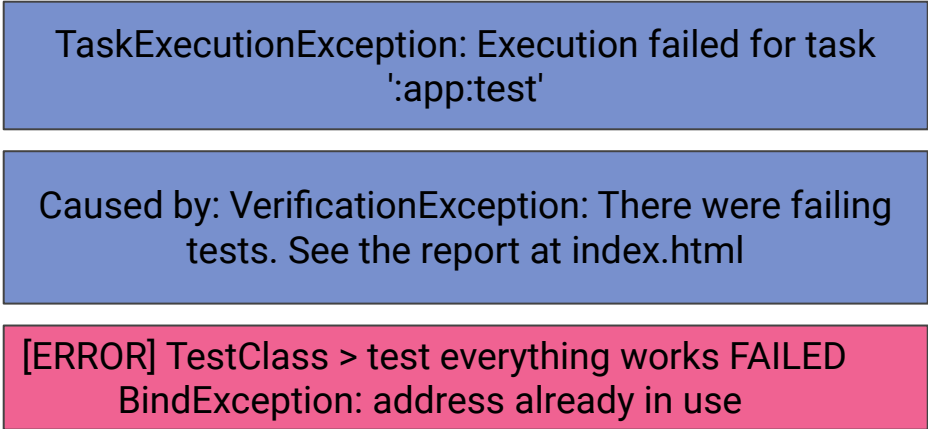
```
TaskExecutionException: Execution failed for task
                        ':app:test'
Caused by: VerificationException: There were failing
           tests. See the report at index.html
```

Log Output ->

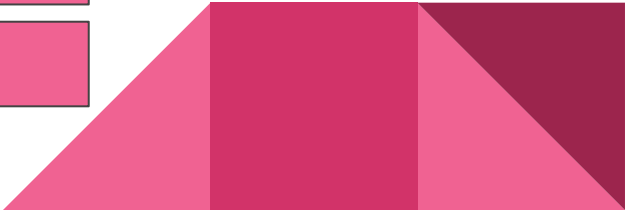
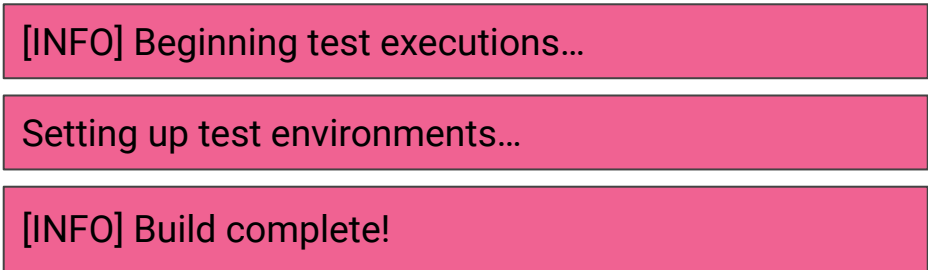
```
[INFO] Beginning test executions...
[INFO] Setting up environment
[ERROR] TestClass > test everything works FAILED
        BindException: address already in use
[INFO] Build complete!
```

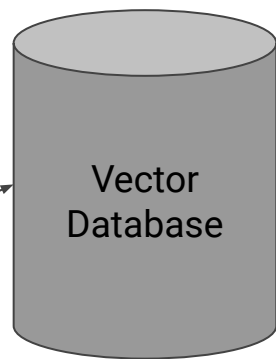

Build: "abcdefg", Failure: #1

Embeddings
to keep ->

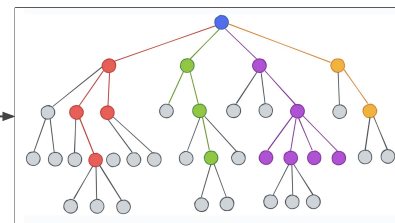


Embeddings
to filter out ->





Classification Model



1. Search

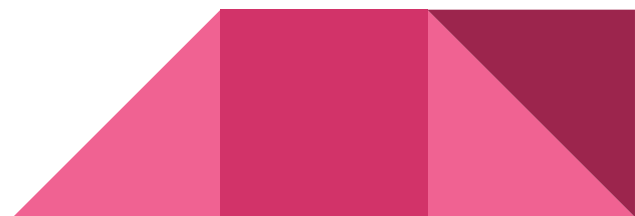
Select all failures that contain failing tests caused by a `BindException`

2. Clustering

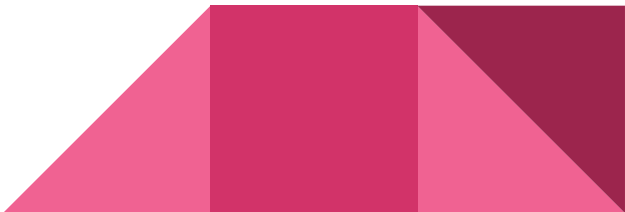
Select all failures that have occurred on CI in the last 3 days, grouped by cosine similarity

3. Classification

Select all failures caused by lint checks failing




Failure embeddings – pitfalls to watch out for

- ❑ Text data is messy. It is easy to confuse the model and create embeddings that overgeneralize if you're not careful.
 - Exception classes can appear in code snippets
 - Error messages can be part of test assertions meant to test error message validity
 - Test classes can appear in text chunks that are describing a compilation of linting error.
 - ❑ Qualitative review is essential.
 - ❑ Good embeddings alone aren't enough, additional logic and user input is sometimes required.
- 



Where to next?

Where to next?

- ❑ Identify how and when to seek expert input
 - ❑ Automatically detect and suggest remediations
 - ❑ Evaluate alternative embeddings generation approaches
 - ❑ Expand training and evaluation data by working with Develocity users
 - ❑ Iterate towards Develocity features
- 

Automating Build Failure Analysis Using Semantic Embeddings

Justin Merkel, Luke Daley