

# **SSW 565 FINAL PROJECT**

## **Option 2**

**Prof. Ying Wang**

**Khusboo Patel**

### **1. Introduction**

Majority of the software systems use issue trackers to report defects, and commits to source control repositories to integrate code changes. As a consequence, detecting vulnerabilities is as easy as reviewing these fundamental artifacts of software development (a new bug report or commit) in real time. Furthermore, source code updates, bug reports, and commit comments typically offer considerable contextual information articulated in basic English that assists a researcher in determining whether the underlying item is associated with a quality attribute. The most common way to evaluate a commit for important insights is to categorize the interactions based on their subjects, bug solutions, and feature additions.

Determining the purpose of a pull request is highly valuable when attempting to enhance goods since it answers the issue of why people are updating the code. However, deciding whether to employ user commit messages is sometimes seen as a classification challenge. As a result, starting will almost always demand a substantial amount of labeled data. Both Microsoft's LUIS and Google's Dialogflow, for example, presume that you can utilize either prebuilt domain labeled data or previously labeled data.

In this article, I'll explain a method for automatically clustering short-text message data in order to identify and extract intents, as well as how to categorize new commits using previously classified data. To automate the identification procedure, I employed natural language processing and machine learning techniques on the acquired dataset, which contains a broad mix of security-related commits and bug reports from the OpenStack project.

### **2. Related Work**

Dynamic analysis examines source code by subjecting it to real-world inputs. In order to discover faults, basic dynamic analysis (or testing) algorithms try a variety of different inputs. There are additional dynamic taint analysis tools available, which track taints in real time.

PHP Aspis, for example, uses dynamic taint analysis to identify XSS and SQL vulnerabilities. ZAP, a frequently used industrial testing tool, identifies particular sorts of internet application vulnerabilities. Users must disable scan policies before doing manual penetration testing.

Static analysis is a technique for evaluating source code or binaries without running them. Examining software written in high-level languages such as C, C++, Perl, PHP, and Python has taken a significant amount of time and effort. These analyzers, on the other hand, are language-

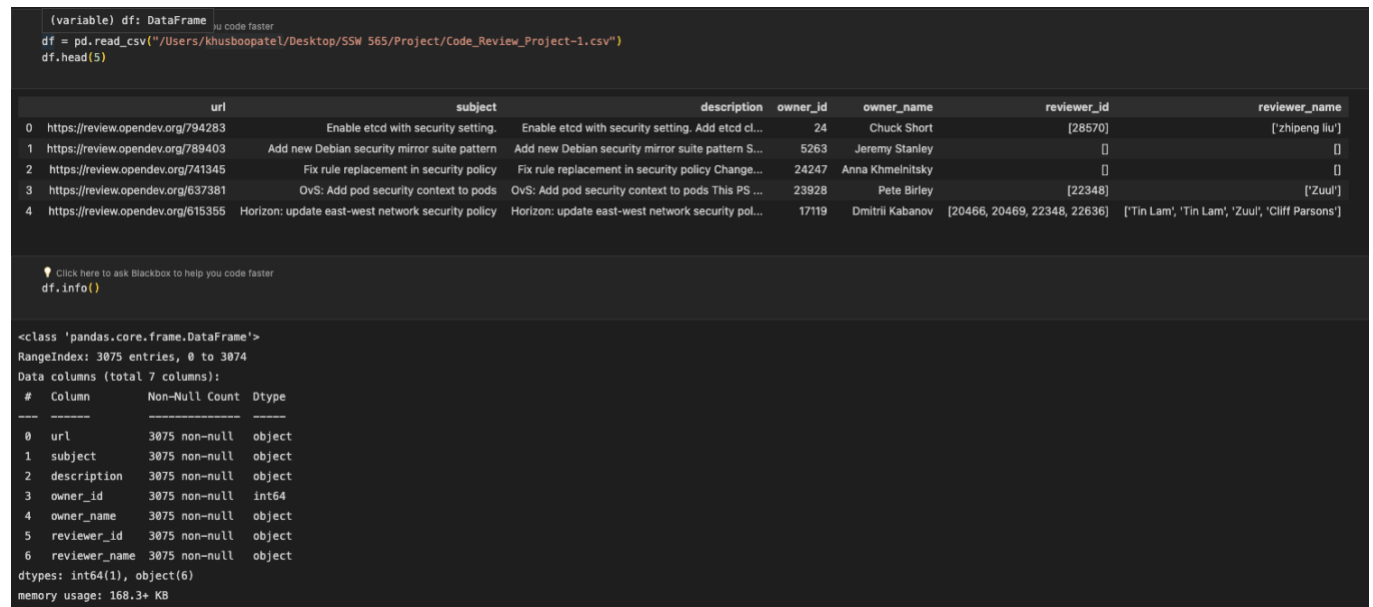
specific and may fail to recognize anything even in languages that are supported. RATS, for example, does not identify weaknesses such as Cross-Site Scripting (XSS) or SQL Injection. Furthermore, when applied to real-world projects, these strategies create a large number of false positives that are difficult to remove.

The works mentioned above are related to discovering new vulnerabilities in source code. But this project is limited to classifying the user commits into categories and classifying new commits into those categories.

### 3. Approach

I now propose the design of an unsupervised machine learning-based commit-message/bug-report-based topic identifier. The identifier retrieves and categorizes the topic from the dataset of commits/bug messages provided in the file. The new commits are then classified into pre-existing categories to facilitate identification.

### 4. Dataset



```
(variable) df: DataFrame
In code faster
df = pd.read_csv("/Users/khusboopatel/Desktop/SSW 565/Project/Code_Review_Project-1.csv")
df.head(5)
```

	url	subject	description	owner_id	owner_name	reviewer_id	reviewer_name
0	https://review.openstack.org/794283	Enable etcd with security setting.	Enable etcd with security setting. Add etcd cl...	24	Chuck Short	[28570]	['zhipeng liu']
1	https://review.openstack.org/789403	Add new Debian security mirror suite pattern	Add new Debian security mirror suite pattern S...	5263	Jeremy Stanley	[]	[]
2	https://review.openstack.org/741345	Fix rule replacement in security policy	Fix rule replacement in security policy Change...	24247	Anna Khmel'nitsky	[]	[]
3	https://review.openstack.org/637381	OvS: Add pod security context to pods	OvS: Add pod security context to pods This PS ...	23928	Pete Birley	[22348]	['Zuul']
4	https://review.openstack.org/615355	Horizon: update east-west network security policy	Horizon: update east-west network security pol...	17119	Dmitrii Kabanov	[20466, 20469, 22348, 22636]	['Tin Lam', 'Tin Lam', 'Zuul', 'Cliff Parsons']

Click here to ask Blackbox to help you code faster  
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3075 entries, 0 to 3074
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0    url         3075 non-null   object
1    subject     3075 non-null   object
2    description 3075 non-null   object
3    owner_id    3075 non-null   int64
4    owner_name  3075 non-null   object
5    reviewer_id 3075 non-null   object
6    reviewer_name 3075 non-null  object
dtypes: int64(1), object(6)
memory usage: 168.3+ KB
```

Figure 1: Information on Openstack commits dataset provided for the project

Because the subject column are too tiny and might be deceptive, the commit description piques our curiosity. The description, on the other hand, offers a lot more information that may be used to guess which category the commits fall under.

## **Topic modeling**

An unsupervised learning problem like this can be approached in a number of ways. When confronted with this situation, the first method that came to mind was topic modeling. It's a method for uncovering hidden subjects in a collection of documents. Understanding which issues or features are being worked on is a critical first step toward automating the process of classifying commits based on quality attributes or security fixes. However, I don't expect to find a perfect solution because I'm using an unsupervised model, and the labelling of commit messages is quite subjective.

Having stated that, here are the steps for creating the model:

- o Step 1: Find the relevant Data
- o Step 2: Remove emails, tokenize words and Clean-up text
- o Step 3: Create Bigram and Trigram Models
- o Step 4: Lemmatize

Topic modeling can be done in a variety of ways, but Latent Dirichlet Allocation (LDA) is one of the most used models. LDA is a probabilistic generative model in which each document is composed of a distribution of a set of subjects, each of which is composed of a set of words. Determining how many topics to use, which is an important model hyperparameter when using LDA (and many other topic modeling approaches), is a key challenge.

## **Clustering the embeddings**

Clustering is a well-known technique for resolving the difficulties with unsupervised learning. To cluster text data, we'll need to make a lot of decisions, including how to process the data and which algorithms to apply.

To begin with, the text data must be numerically represented in order for the computer to operate on it. One method for clustering is to create embeddings, or vector representations, of each word. Because each message contains multiple words, one option is to simply average the individual word embeddings of each message's words.

There are several approaches to determining a single vector representation of a sentence. Google's Universal Sentence Encoder (USE), first published in 2018 by Cer, is a popular sentence embedding model. The USE model was trained on a diverse set of data, including Wikipedia, web news, web question-and-answer pages, and discussion forums, and it excels at task of sentence semantic similarity.

In a publication released in 2019, Reimers and Gurevych proposed Sentence-BERT, a "modification of the pretrained BERT network that uses Siamese and triplet network architectures to create semantically relevant sentence embeddings that can be compared using

cosinesimilarity." There are several Python implementations of these models trained on various datasets, which are available to download and use from HuggingFace.

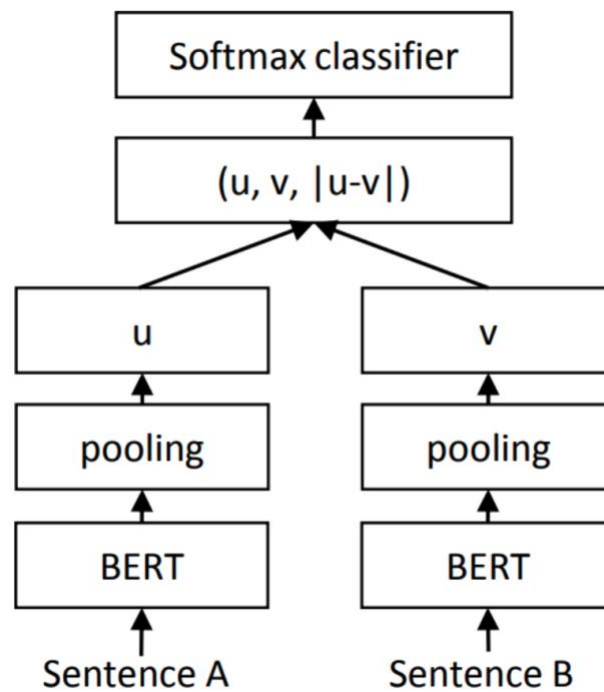


Figure 2: SBERT architecture with classification objective function


It is apparent from the image that BERT makes up the base of this model, to which a pooling layer has been appended. The pooling layer enables us to create a fixed-size representation for input sentences of varying lengths. The image shows the base architecture and objective function together which creates the SBERT model.

The Scikit-learn documentation gives a good overview of the various clustering algorithms that it supports and when each performs best. It is preferable for our current application to use an algorithm that does not require the number of clusters to be specified in advance and can also tolerate noisy data. Density-based algorithms are ideal because they do not require the number of clusters to be specified and are unconcerned about cluster shape. Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) has gained popularity because it has fewer and more intuitive hyperparameters and is resistant to variable-density clusters. A comparison of various clustering algorithms is included in the HDBSCAN documentation. HDBSCAN worked best for similar kind of problem, so we'll use it for the rest of the project.

There are two popular packages available for linking UMAP and HDBSCAN for topic modeling: Top2Vec and BERTopic. The default hyperparameters in both packages, however, do not perform well for problems with a small corpus, such as the current one (most of the data ends up being classified as noise).

## 5. RESULTS

Manually examining how the models performed on some of the ground truth clusters gives us the following results:



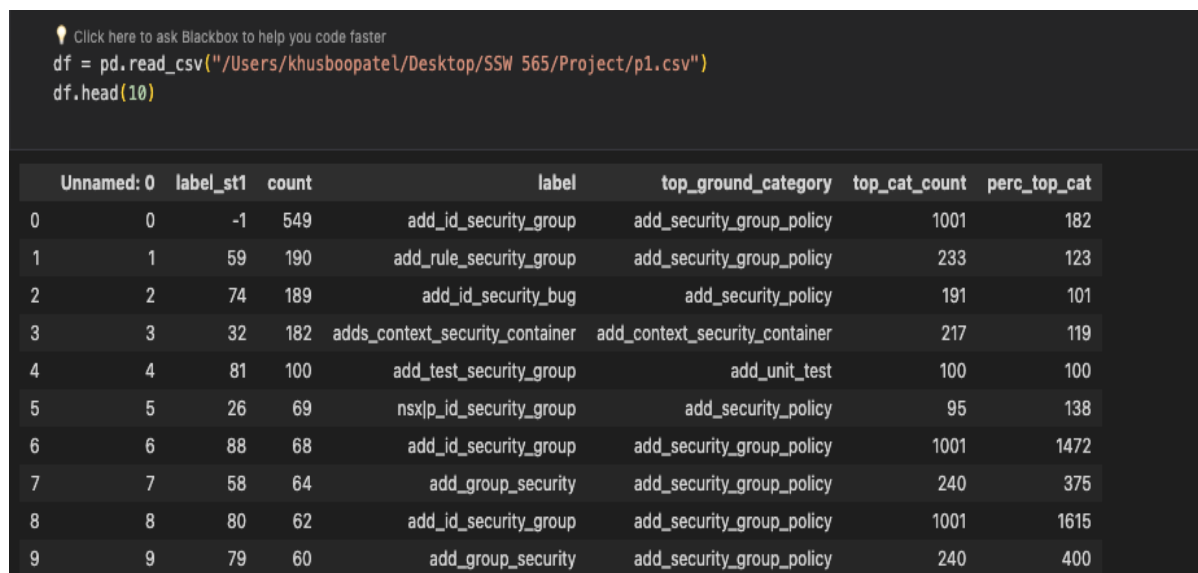
```
Click here to ask Blackbox to help you code faster
df = pd.read_csv("/Users/khusboopatel/Desktop/SSW 565/Project/compare_stats.csv")
df.head(5)
```

	Model	ARI	NMI	loss	label_count	n_neighbors	n_components	min_cluster_size	random_state
0	st1	0.171	0.568	0.180813	90	3	6	8	42
1	st2	0.052	0.301	0.209106	89	12	11	6	42
2	st3	0.145	0.179	0.179919	4	8	7	15	42
3	use	0.112	0.149	0.164309	4	9	8	13	42

Figure 3: Statistical Analysis of the models

The Adjusted Rand Index (ARI) is frequently used in cluster validation since it is a measure of agreement between two partitions: one given by the clustering process and the other defined by external criteria.

Normalized mutual information (NMI) gives us the reduction in entropy of class labels when we are given the cluster labels. In a sense, NMI tells us how much the uncertainty about class labels decreases when we know the cluster labels. It is like the information gain in decision trees.



```
Click here to ask Blackbox to help you code faster
df = pd.read_csv("/Users/khusboopatel/Desktop/SSW 565/Project/p1.csv")
df.head(10)
```

	Unnamed: 0	label_st1	count	label	top_ground_category	top_cat_count	perc_top_cat
0	0	-1	549	add_id_security_group	add_security_group_policy	1001	182
1	1	59	190	add_rule_security_group	add_security_group_policy	233	123
2	2	74	189	add_id_security_bug	add_security_policy	191	101
3	3	32	182	adds_context_security_container	add_context_security_container	217	119
4	4	81	100	add_test_security_group	add_unit_test	100	100
5	5	26	69	nsxjp_id_security_group	add_security_policy	95	138
6	6	88	68	add_id_security_group	add_security_group_policy	1001	1472
7	7	58	64	add_group_security	add_security_group_policy	240	375
8	8	80	62	add_id_security_group	add_security_group_policy	1001	1615
9	9	79	60	add_group_security	add_security_group_policy	240	400

Figure 4: Comparison of the labels generated by the model with my labels

Click here to ask Blackbox to help you code faster

```
df = pd.read_csv("/Users/khusboopatel/Desktop/SSW 565/Project/models_labels_results.csv")
df.head(15)
# print(df)
```

Unnamed: 0		text	category	label_use	label_st1	label_st2	label_st3
0	0	Enable etcd with security setting Add etcd cli...	add_context_security_etcd	2	29	-1	1
1	1	Add new Debian security mirror suite pattern S...	add_security_mirror_repository	2	55	34	2
2	2	Fix rule replacement in security policy Change...	add_security_policy	2	71	41	2
3	3	OvS: Add pod security context to pods This PS ...	add_context_security_container	1	32	11	1
4	4	Horizon: update east-west network security pol...	add_security_group_policy	2	59	29	2
5	5	Implement helm-toolkit snippet to Keystone pod...	add_context_security_container	1	32	11	1
6	6	Add missing security context to Mini-mirror te...	add_context_security_container	1	3	11	1
7	7	Enable setting default rules for default secur...	add_security_group_policy	2	58	43	2
8	8	Testingrst misses a step for adding security r...	add_unit_test	2	81	-1	2
9	9	Test security groups Install ovs kernel module...	add_security_group_policy	2	35	-1	2
10	10	Add OSSN-0040 security groups don't work with ...	add_security_group_policy	2	35	68	2
11	11	Add security groups dscp spec This adds the sp...	add_security_group_policy	2	64	-1	2
12	12	Open vSwitch-based Security Groups: OVS Firewa...	add_security_group_policy	2	35	68	2
13	13	Add security groups for nova As current implem...	add_security_group_policy	2	58	72	2
14	14	Add security to libvirt Allow libvirt to be a...	add_unit_test	2	13	8	-1

Figure 5: Comparision of the labelling done by the models

For problem statement p1, I proposed a technique for employing domain knowledge to generate a restricted optimization problem that can automatically adjust UMAP and HDBSCAN hyperparameters. This allows us to simply organize code commits and identify them descriptively.

Before making a decision or needing to perform time-consuming manual labeling, the clustering findings give helpful insights into unlabeled text data in a very short amount of time. This eventually becomes a question of grouping the clusters depending on the keywords that we are looking for.

## References:

1. <https://github.com/pandas-profiling/pandas-profiling>
2. <https://docs.microsoft.com/en-us/azure/cognitiveservices/luis/luis-concept-intent>
3. <https://cloud.google.com/dialogflow/es/docs/intents-overview>
4. <https://www.acsac.org/2000/papers/78.pdf>
5. [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)
6. <http://dl.acm.org/citation.cfm?id=2002168.2002170>
7. <https://dl.acm.org/doi/10.1145/360248.360252>
8. <https://www.usenix.org/events/osdi08/tech/full>
9. <https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods>
10. [https://en.wikipedia.org/wiki/Topic\\_model](https://en.wikipedia.org/wiki/Topic_model)
11. <https://medium.com/pew-research-center-decoded/making-sense-of-topic-models-953a5e42854e>
12. <https://huggingface.co/>
13. <https://towardsdatascience.com/how-to-cluster-in-high-dimensions-4ef693bacc6>