BOOKHIVE LIBRARY MANAGEMENT SYSTEM (LMS)

WWW Structures, Techniques and Standards
CSI 3140

Dynamic Code Analysis Report
Phase 3

Team #33

Student Number	Name	Email
300163562	Lara Lim	llim067@uottawa.ca
300173889	Amani Farid	afari094@uottawa.ca
300184721	Samy Touabi	stoua026@uottawa.ca
300098986	Kian Zahrai	kzahr091@uottawa.ca

Date Submitted: July 23rd, 2023
Professor: Khalil Abuosba

Introduction

Dynamic code analysis, also known as runtime code analysis or dynamic testing, is an essential technique used in software development to evaluate and analyze code behavior while it is running. It involves monitoring the code execution during runtime to identify potential issues, bugs, and vulnerabilities. Dynamic code analysis is particularly valuable because it allows developers to gain insights into how their code behaves under real-world conditions and interactions.

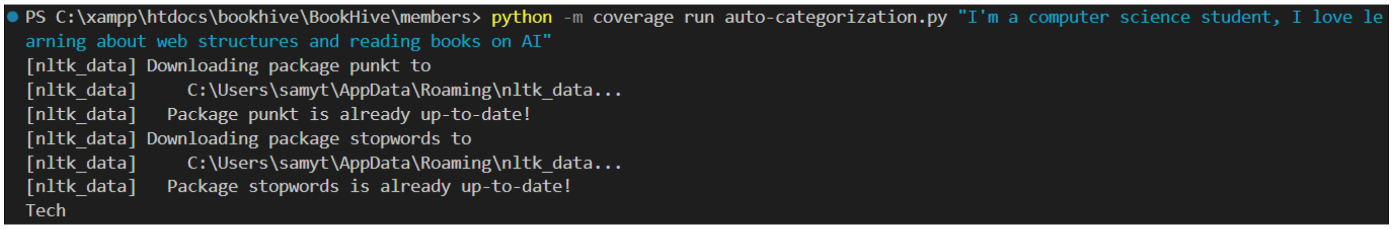
Benefits of using dynamic code analysis:

1. **Bug Detection:** Dynamic code analysis helps identify bugs, errors, and unexpected behaviors in the code during runtime, which may not be apparent during static analysis or code reviews.
2. **Test Coverage:** It provides visibility into the code paths that are executed during testing, helping to assess test coverage and identify untested or under-tested sections.
3. **Performance Optimization:** By analyzing code behavior in real-time, developers can pinpoint performance bottlenecks and optimize code for better efficiency.
4. **Security Testing:** Dynamic code analysis can help identify security vulnerabilities and potential risks, such as buffer overflows, SQL injection, and cross-site scripting, by analyzing how the code handles input and data.
5. **Integration Testing:** It aids in integration testing, ensuring that different components of the code work together seamlessly and produce the desired results.
6. **Continuous Integration:** Dynamic code analysis can be integrated into the continuous integration and continuous delivery (CI/CD) pipeline, automating the process of identifying code issues as part of the development workflow.
7. **Error Diagnosis:** It assists in diagnosing the root cause of runtime errors and exceptions, providing valuable information for debugging and troubleshooting.

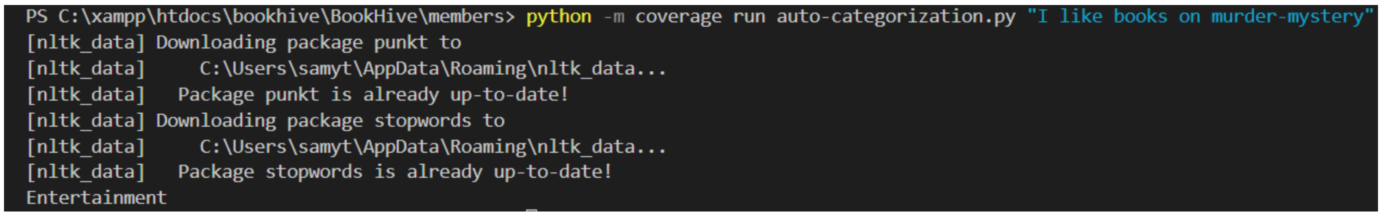
In summary, dynamic code analysis is a powerful tool that complements other software testing and quality assurance practices. It helps developers ensure code reliability, security, and performance by observing its actual behavior during runtime, ultimately leading to the delivery of more robust and reliable software products.

For this project, we have used Python’s coverage module to test our Natural Processing Language tool for Sentimental Analysis, which we built with Python. Python's coverage module is used for dynamic code analysis to measure the code coverage during runtime testing. It helps developers assess how much of the code is being executed and tested, providing insights into the effectiveness of the test suite and identifying areas that might be lacking proper testing.

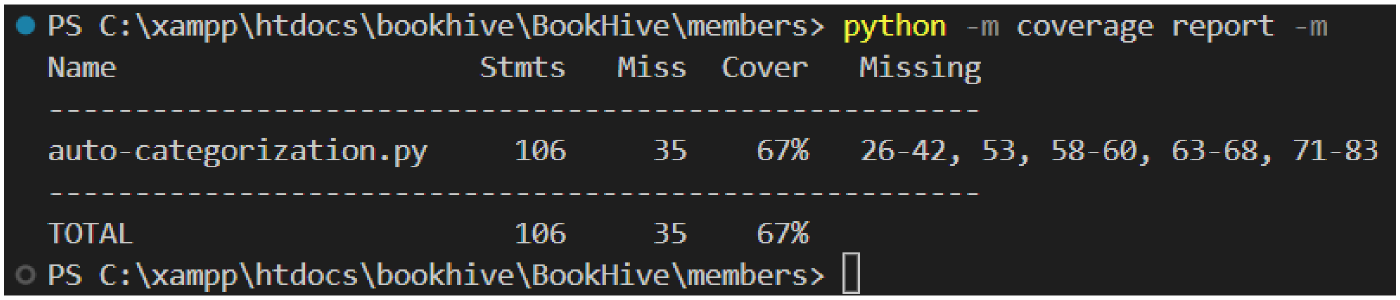
Here a few screenshots of the results upon testing:



```
PS C:\xampp\htdocs\bookhive\BookHive\members> python -m coverage run auto-categorization.py "I'm a computer science student, I love le
arning about web structures and reading books on AI"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\samyt\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\samyt\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Tech
```



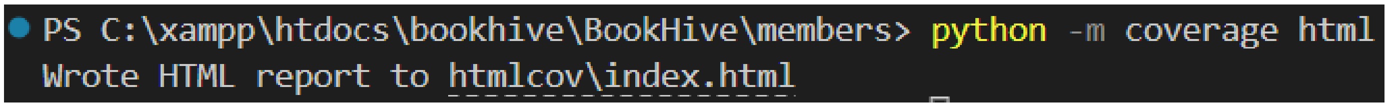
```
PS C:\xampp\htdocs\bookhive\BookHive\members> python -m coverage run auto-categorization.py "I like books on murder-mystery"
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\samyt\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\samyt\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Entertainment
```



```
PS C:\xampp\htdocs\bookhive\BookHive\members> python -m coverage report -m
Name                        Stmts   Miss  Cover   Missing
---------------------------------------------------------
auto-categorization.py        106     35    67%   26-42, 53, 58-60, 63-68, 71-83
---------------------------------------------------------
TOTAL                         106     35    67%
PS C:\xampp\htdocs\bookhive\BookHive\members>
```

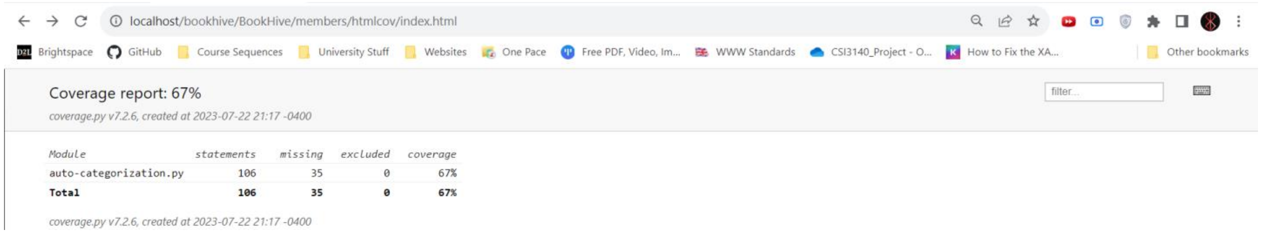
With just a few tests, we achieved 67% Coverage, which is quite good.

Python’s Coverage module also allows users to neatly view which part of the code has been covered and which has not been through an HTML report.



```
PS C:\xampp\htdocs\bookhive\BookHive\members> python -m coverage html
Wrote HTML report to htmlcov\index.html
```

Below are some screenshots of the report, which we also made available in the repository (in the file seen above).

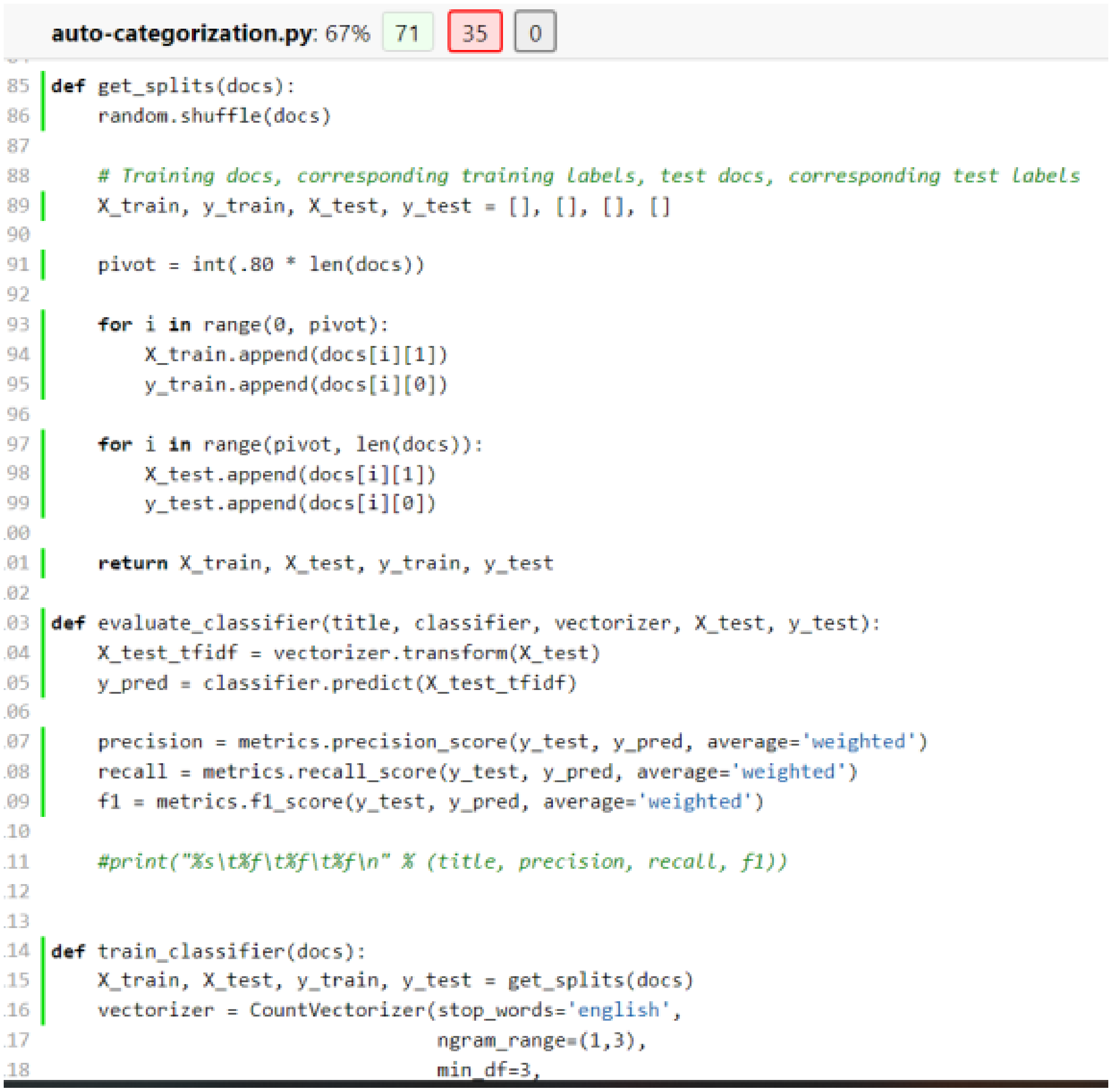


Coverage report: 67%

coverage.py v7.2.6, created at 2023-07-22 21:17 -0400

Module	statements	missing	excluded	coverage
auto-categorization.py	106	35	0	67%
Total	**106**	**35**	**0**	**67%**

coverage.py v7.2.6, created at 2023-07-22 21:17 -0400



auto-categorization.py: 67% 71 35 0

```
def get_splits(docs):
    random.shuffle(docs)

    # Training docs, corresponding training labels, test docs, corresponding test labels
    X_train, y_train, X_test, y_test = [], [], [], []

    pivot = int(.80 * len(docs))

    for i in range(0, pivot):
        X_train.append(docs[i][1])
        y_train.append(docs[i][0])

    for i in range(pivot, len(docs)):
        X_test.append(docs[i][1])
        y_test.append(docs[i][0])

    return X_train, X_test, y_train, y_test

def evaluate_classifier(title, classifier, vectorizer, X_test, y_test):
    X_test_tfidf = vectorizer.transform(X_test)
    y_pred = classifier.predict(X_test_tfidf)

    precision = metrics.precision_score(y_test, y_pred, average='weighted')
    recall = metrics.recall_score(y_test, y_pred, average='weighted')
    f1 = metrics.f1_score(y_test, y_pred, average='weighted')

    #print("%s\t%f\t%f\t%f\n" % (title, precision, recall, f1))


def train_classifier(docs):
    X_train, X_test, y_train, y_test = get_splits(docs)
    vectorizer = CountVectorizer(stop_words='english',
                                 ngram_range=(1,3),
                                 min_df=3,
```