**3. METHODOLOGY**

To collect machine learning-specific code smells, we resort to academic literature, grey literature, community-based coding Q&A platforms (with Stack Overflow), and public software repositories (with GitHub).

**3.1 Paper Mining :-**

**1)** Search in google scholar engine for collecting potentially code smells ML papers. Used terms containing machine learning related keywords and cod quality related keywords to search.

**2) Selection based on title and abstract:**

Some papers are ML for S/W engineering

Some papers are S/W Eng for machine Learning.

When we can not classify a paper from it’s title they look into abstract and decide whether to include or not

**3) Snowballing:-** apply forward and backward snowballing method. Browse reference list of 33 papers and the list where the paper is cited, select paper based on paper and abstract. Delete duplicated papers and 9 papers were added after this.

**4) Full-text reading and selecting the ones with potential machine learning-specific code smells:-**

Read full text of 42 papers, and select one wth potential machine learning specific code smells. After that they got 6 final papers that contribute to the code smell catalog.

**3.2 Grey Literature Mining**

In many cases ML knowledge is published on the web by experienced practitioners. For example blog posts. Inorder to collect online entries we make use of google search engine with the same queries used above for the research literature(cf. fig3).

6 most famous ML libraries combine them with the code quality related keywords and form new group of search queries covering the two most important steps in machine learning application development – data processing and model training. Not all entries contain actionable coding knowledge so we use 1) reading the title, 2) reading the first summary, and 3)

reading the whole article to select specific once.

**3.3 Reusing Existing Bug Datasets**

We reuse the dataset provided in the work by Zhang et al. [22] to mine code smells in Tensorflow applications. Zhang et al. mined the Tensorflow application bugs, analyzed the bugs pattern using 88 Stack Overflow posts as well as 87 GitHub commits and provided a replication package for these bugs (hereinafter called “TensorFlow Bugs” replication package). We reuse their replication package to extract recurrent pitfalls that may generalize to other projects and thus should be documented as code smells.

**3.4 Complementary Stack Overflow Mining**

TensorFlow has already checked stacker overflow and GitHub commits. In this method we only check in stack over flow for this reason to avoid duplication. 3 step

1. Library Selection, 1) Library Selection, 3) Applying Search Terms

**3.5 Validation**

First author collects all code smells from empirical study and discuss with second author. First author collected 31 code smells, from which 9 were dropped.

**4 RESULTS**

22 ML codes collected. General description followed by context of smell, the problem of it’s occurrence and the solution. Summarizes all smells including references supporting the smell.’

The stage of ML pipeline where they are more relevant and the main effect that arises from having those smells.

**4.1 Unnecessary Iteration**

Avoid unnecessary iteration. Use vectorized solutions instead of loops.

Context: time consuming loops.

Problem : Iteration through pandas objects is generally slow. In most cases iteration manually over rows not needed and can be avoided. slicing operation with loops in TensorFlow is slow, and there is a substitute for better performance

Solution: ML applications are usually data intensive so better use vectorized method instead of doing iterations on data. Thus program runs faster. Pandas’ built-in methods (e.g., join, groupby) are vectorized. In TensorFlow, using the tf.reduce\_sum() API to perform reduction operation is much faster than combining slicing operation and loops.

**4.2 NaN Equivalence Comparison Misused**

Context: NaN enqualance comparison different behave differently from normal equilance comparison.

Problem : While None == None evaluates to True, np.nan == np.nan evaluates to False in NumPy

Pandas treat None as np.nan so comparison of Dataframe with np.nan always return false. If developer is not aware of this, this could lead to unconditional behaviors in the code.

Solution: developers need to be careful using np.nan for comparison

**4.3 Chain Indexing**

Context: In Pandas, df[“one”][“two”] and df.loc[:,(“one”,“two”)] give the same result. df[“one”][“two”] is called chain indexing. Problem Using chain indexing may cause performance issues

Problem: chain indexing causes performance issues as well as error-prone code.

df[“one”][“two”], Pandas sees this operation as two events

df.loc[:,(“one”,“two”)] only performs

a single call. In this way, the second approach can be significantly

Solution: developers avoid using chain indexing.

**4.4 Columns and DataType Not Explicitly Set**

Context: when file is imported to DF , datatypes of columns are set by default dtype conversion.

Problem: If columns are not set with correct datatype developers do not know what to expect in the downstream data schema. Also if datatyoe is not set properly, it may silently continue to next step even if the input is unexpected.

Solution : set column datatype accordingly.

**4.5 Empty Column Misinitialization**

Context: When new ocolumns added in pandas columns are added with nan’s instead of zeros or empty strings.

Problem: If they use zeros or empty strings to new empty column in the Dataframe it may also happens to initializations in other data structure or libraries.

Solution : Use NaN values (np.nan) if a new columns is needed in a Dataframe. Do not use “filter values such as zeros or empty strings.

**4.6 Merge API Parameter Not Explicitly Set**

Context: df.merge() API merges two DataFrames in Pandas

Problem: merge operation is computationaly memory expensive, the parameter on states which columns to join on. Also describes how the join (inner, left). Also the validate parameter will check wether the merge is of a specified type. It is preferable to do the merging process in one stroke for performance consideration.

Solution : Developer should explicitly specify the parameters for merge operation.

**4.7 In-Place APIs Misused**

Context: data structure changed 2 types. Take a copy of data and change the structure on copy

Second is change the existing data structure(also known as in place).

Problem: some times changing values to the existing Dataframe wil not be stored to a variable, in this case . Some developers use df.dropna() to original df assuming that it’s saved on the Dataframe without using the inplace=true.

Solution : developers check weather the result of an operation assigned to a variable. Some develpers think that inplace parameter save momory, but it’s a misconception about pandas, anyway system is creating a copy of the Dataframe so this is a different scenario.

**4.8 Dataframe Conversion API Misused**

Context: Pandas, df.to\_numpy() and df.values() both can turn a DataFrame to a NumPy array.

Problem:

Solution :

Context:

Problem:

Solution :

Context:

Problem:

Solution :

**5. Discussion and Implication**

Machine Learning catalog summarized from Empirical studies

Collected 22 code smells in total and linked them to four pipeline stages: Data Cleaning, Feature Engineering, Model Training, Model Evaluation.

16 smells are generic and 6 smells are API-specific smells. Generic smells occur no matter which library the user uses. On the other hand API-Specific smells depends on the specific library API design.

Possible Impacts of smells on application codes are : error-prone, less efficient, less re-producible, causing memory issues, less readable and less robust.

Most of the times data scientists would not have software Engineering background and are not up-to-date with the best practices from the software engineering field. So our catalog of smells mitigate this by providing guidelines during the development of machine learning applications.

Using Tensor Flow bugs, replication package replication package and found that many instances have already been deprecated because TensorFlow has upgraded to version 2. Hence, we expect that new API-specific code smells will appear with new versions and library features. In fact, our results showcase that most API related smells are only reported by grey literature in general instead of literature. We argue that collecting a catalog of code smells helps in promoting a continuous effort between practitioners and academics.

three smells can be considered temporary smells: Dataframe Conversion API Misused, Matrix Multiplication API Misused and Gradients Not Cleared before Backward Propagation.

Study focused on six python ML Libraries and frameworks.

**5.1 Implication to data scientists:** - Unnecessary Iteration code smell describes the inefficient code structure, and it often occurs at data cleaning stages. Another code smell Hyperparameter Not Explicitly Set indicates irreproducible code and it is at model training stage. Data scientists and machine learning application developers can check these aspects while checking their code. Future work will validate whether eliminating these code smells will lead to more accurate results during training, better hyperparameter optimization, clearer and higher quality code, and less maintenance effort.

**5.2 Implication to Machine Learning Library Developers:-** The effect of index chaining appears to be in the examples in stack overflow even though they are explained in the pandas documentation. This indicates that many developers are struggling to follow the documentation strictly. It might stem from the fast iteration cycle in the development process of the team or developer’s lack 0f experience. Author argued that passively showing warning on documentation might not suffice. It is important that library developers has to actively engage in community forums such as stack overflow to help the community avoid non-obvious issues. Hence their contribution is crucial in the development of coding tools that support best practices. Also it’s important for library developers to reach out to existing projects that aim at helping the development of machine learning software tools. Library developers know better than anyone what is optimal way of leveraging their libraries.

**5.3 Implication to Code Analysis Tool Developers :-** As some code smells can not be addressed by designing better API’s. The static analysis tools can be used to follow the best practices and warn the pitfalls to the application developers.

This research serves as the base for providing automated tools for identifying unwanted patterns in the code. Automated tools can minimize the job of developers and providing support for good quality code assurance. Because humans are occasionally forgetful. It is preferable to have technology that expressly checks whether best practices are bring followed.

Furthermore, some code smells are related to the context. This is aligned with previous work that proposes context-aware code analysis tools for machine learning applications. For example, PyTorch library developers recommend application developers to use the deterministic option during the development but not set it in the production code due to the consideration for performance. Therefore, the automated tool can have different configuration settings. For example, according to the pipeline stage, it can have a development setting and a deployment setting.

**5.4 Implication to Students:- Implication to students :-** Many students in the ML industry do not have prior education on machine learning application development since it requires a combination of software engineering and data science practices. Students can make use of this catalog to learn more about the common anti-patterns in machine learning application development and prepare for future jobs.