Mousetraps: Reliably distinguishing free apps and trial versions of paid macOS apps on public app marketplaces

Zakhar Litvinchuk[1] z.litvinchuk@ukma.edu.ua
Oleksandr Frankiv[1] o.frankiv@ukma.edu.ua
Yevhenii Peteliev[2] zhenya.peteliev@macpaw.com
Sergii Kryvoblotskyi[2] krivoblotsky@macpaw.com
Nataliia Stulova[2] nata.stulova@macpaw.com

[1]National University of Kyiv-Mohyla Academy, Kyiv, Ukraine
[2]MacPaw, Kyiv, Ukraine

Abstract

Background. Striving to achieve the ultimate transparency and clear labelling of macOS apps in the public marketplaces is essential for building user trust and making informed decisions.
Aim. This research is aimed at distinguishing truly free apps from those that contain in-app purchases in different forms, and their further classification into free, freemium, and paid classes.
Methods. The analysis was conducted using a combination of technical and language-based approaches to identifying the key features that indicate each particular class. The final classification was performed using a supervised machine learning algorithm.
Results. Among the classifiers we evaluated, Random Forest and Histogram-Based Gradient Boosting emerged as the best-performing models for monetization category classification. Both achieved 90% precision, 90% recall, 89% F1-score, and 90% accuracy.
Conclusions. The findings of this research promote the improvement of automated app classification systems and can be applicable in the software distribution ecosystem.

1 Introduction

Alongside with the evolution of creating applications, companies also began to develop different methods of monetization that aimed both at optimizing user experience and generating maximum revenue. As a result, monetization strategies started to be categorized into distinct classes.

The first monetization model was based on paid applications and involved a one-time purchase of the software. The second monetization model served as an extension of the first and introduced the use of trial periods, allowing potential users to test the application before making a purchase. To address the issue of high entry barrier, the concept of subscription-based monetization was introduced, i.e. involving periodic payments for using the product. However, to encourage the customers to download an app, free use could become the major enticement. At the same time, even this option left space for unlocking additional features or content on a paid basis. The described model of monetization obtained the name "freemium", which is compounded of "free" and "premium". For developers who chose to keep all functionality of the application free, the primary source of revenue became the integration of advertisements.

However, it is common for app developers to avoid clear indications of functionality that depends on in-app purchases or subscription requirements. The potential range of the addressed issue can be represented by the most popular marketplaces: Mac App Store[1], which features about 1.8 million apps currently, and MacUpdate[2], offering more than 32,000 apps and serving over one million active users. After installation, users may discover that access to key features, or even the entire functionality, is restricted unless a payment is made. This approach misleads users, creates poor user experience and undermines their trust.

This pressing problem substantiated the purpose of this research which is aimed at designing an algorithm for identifying monetization models used in macOS applications. Herein, we focused our attention on the Apple ecosystem and on the desktop apps that are distributed on the Mac App Store (MAS) and the MacUpdate marketplaces, and their subsequent classification. To address this, we have developed a solution that combines static analysis of Mach-O binaries with NLP-based processing of text content.

The main contributions of this paper are: (1) an approach that integrates features extracted from Mach-O binaries of application and embedded frameworks with purchase records from MAS re-

[1]https://www.apple.com/app-store/
[2]https://www.macupdate.com/about

ceipt file; (2) extension of the technical analysis that applies natural-language processing to local text components and external textual information from marketplaces including description, user reviews and legal documents (e.g., Privacy Policy and Terms of Use).

We evaluated our solution on the test dataset of 1,219 macOS apps from the Mac App Store and MacUpdate using 5 machine learning algorithms: Decision Trees, Support Vector Machines, K-Nearest Neighbors, Naive Bayes, and Gradient Boosting. The best results in precision, recall, F1 score, and accuracy metrics were obtained with Random Forest and Histogram-Based Gradient Boosting classifiers.

2 Background

A macOS application consists of two main components: a bundle and a respective sandbox. A macOS application bundle [3] is a package that contains all the necessary components for the app to function properly. To end users, it looks like a single file with the extension .app, but in reality, it is a directory that provides a clear hierarchy and ensures that all components are in the appropriate subdirectories.

The main element of the app bundle is the Contents directory, which organizes key subdirectories and files. Its macOS subdirectory contains the application executable file in Mach-O format [4] with the program's machine code. It consists of metadata and segments, which contain sections and load commands —- the instructions for correctly loading these elements into the RAM. Dynamic frameworks and libraries are stored in the Frameworks directory.

The Info.plist file is a property list file in XML format, which contains key-value pairs with metadata required for the system to properly launch and manage the app. The Resources directory contains static assets such as images, storyboards, sound and video files, localizations etc. If the application was downloaded from the Mac App Store, the bundle also contains a _MASReceipt subdirectory with a receipt [5] file that stores data about purchases already made and active/inactive subscriptions.

A sandbox [6] is an isolated container within which an application runs. It serves as an isolated file system and restricts access to other files and processes of other applications. The sandbox is created automatically when the application is downloaded and is stored along with the containers of other applications. The default components of the sandbox are UserDefaults and Logs. UserDefaults are property list files for saving configuration values between program launches. Logs are programmatically created structured event records that can be stored in completely different formats.

3 Methodology

Software analysis is traditionally based on static or dynamic analysis (Samarasekara et al., 2023). However, to extend the information about an app, an approach was applied that didn't require a source code or an installed application. It can be called external analysis. A combination of approaches, which were elements of static and external analysis, was used for the algorithm design. It was divided into three phases:

1. Identify technical features related to in-app purchases via bundle components analysis.
2. Identify language features:
   a) extract and process app-related textual content retrieved from bundle and sandbox components;
   b) retrieve external textual content and its further processing.
3. Classify applications based on identified features into *free*, *freemium*, and *paid*.

4 Algorithm overview

Figure 1 shows the complete algorithm pipeline, in which each phase relies on a distinct set of features.

4.1 Technical analysis

Technical analysis was based on examining such bundle components: the app's binary, frameworks and MAS receipt file.

App binary. Payment frameworks, along with classes and functions, were extracted from the app's binary. Apple almost always requires the use of StoreKit framework for integrating in-app purchases for the apps distributed through the Mac App Store; otherwise, the app will be rejected during the app review [7]. However, such a requirement is not applicable to other marketplaces. In this case,

[3] Bundle
[4] Mach-O Runtime Reference
[5] Mac App Store Receipt
[6] Sandbox
[7] App Store Review Guidelines

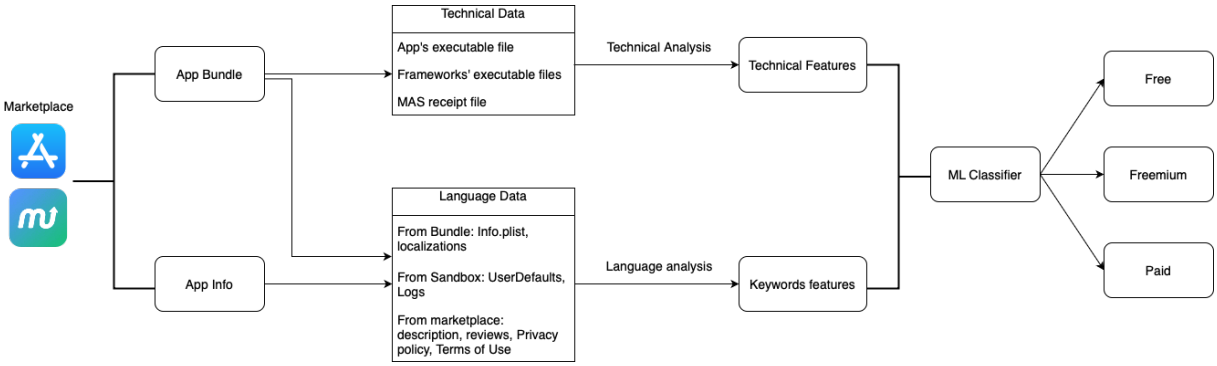

Figure 1: Algorithm pipeline

payments can be integrated using the third-party payment providers in three different ways: through the SDK, by making requests to a REST API, or via checkout web pages. URLs for endpoints and pages used to process or confirm transactions are stored in the "__cstring" section, which belongs to the "__TEXT" segment of Mach-O binary. This section stores all string constants used in the code. Their extraction was based on matching all strings with regular expressions describing the addresses of the most popular payment services.

App frameworks In addition to analyzing the app's binary file, a similar analysis was also performed on the executable files of frameworks used by the application. This allows to identify the frameworks that serve as wrappers for more convenient interaction with the system payment API.

Receipt file In older versions of macOS, a receipt file was used to store information about purchases and subscriptions, and their subsequent local or remote validation [8]. Thus, the algorithm involved decrypting the receipt file and extracting records about the completed purchases, active and inactive subscriptions.

4.2 Language analysis

Language analysis was based on extracting the keywords related to in-app purchases. Searching for keywords required text preprocessing performed through lemmatization, i.e., reducing morphological variants to one dictionary base form (Plisson et al., 2004).

4.2.1 Local text content

To collect the local text app-related content, following bundle and sandbox components were selected: from bundle — Info.plist and localizations, from sandbox — UserDefaults and logs. Both Info.plist and UserDefaults store data in XML in key-value format. Key naming followed CamelCase style (Binkley et al., 2009), thus to extract separate words, each key identifier was split at points where letter case changed.

4.2.2 External text content

External analysis was grounded on retrieving information about the app from public marketplaces, Mac App Store and MacUpdate. Considering the fact that app's description and user reviews could also contain in-app purchases related keywords, they were involved in external analysis. Besides, this stage of analysis was also applied to legal documents such as Privacy Policy and Terms of Use.

Privacy policy is intended to protect sensitive user data, which also includes payments data, e.g. what data is collected and how, how it is stored, for what purpose it is used, and whether the data is shared with third parties, such as payment services for transaction processing.

Terms of Use is a tool to protect companies' copyright and can contain information about the payment terms, refund period, the currency in which the payment is made, information about additional charges such as taxes or service fees, and penalties in case of a late payment (Robinson and Zhu, 2020).

Unlike the terms of use, which are not obligatory, Apple requires that all apps include links to the privacy policy. These links are available on the apps' Mac App Store web page. Links to Terms of Use are usually included in the app description.

[8] Choosing a Receipt Validation Technique

5 Data collection

In this work, we have focused on two aforementioned app marketplaces in the Apple ecosystem — the official Mac App Store (MAS), and the largest unofficial marketplace, MacUpdate.

5.1 App Info Collection

iTunes API To retrieve app description and user reviews from MAS, the iTunes Search API [9] was used. The API response also includes a link to the app's Mac App Store webpage, the content of which was used for web scraping.

Web scraping Web scraping was employed to obtain Terms of Use and Privacy Policy from the Mac App Store, as well as app descriptions and user reviews from MacUpdate. The process targeted two content types: static (app description, Privacy Policy, and Terms of Use), which were extracted directly from HTML pages, and dynamic (paginated user reviews), which normally require browser interaction. The link to Privacy Policy was extracted by searching for a specific HTML tag, while the Terms of Use link was obtained from the app description using a generic URL pattern. To avoid launching a browser for retrieving dynamic content, the underlying network requests for loading additional review pages were analyzed and reproduced. Once the links to the Privacy Policy and Terms of Use were identified, all their text content was scraped for the subsequent language analysis.

5.2 Dataset

Here, we have worked with a share of the "macOS Applications Bundles and Metadata" dataset[10], originally containing over 24k apps, collected from our target marketplaces. For selecting the applications that are popular among users and limiting the inclusion of abandonware, we applied the following criteria:

- each app must have at least 10 ratings
- each app must have at least 10 user reviews
- each app must have been updated within one year of the dataset creation date (July 1, 2024).

We additionally limited ourselves to apps belonging to categories of *Utilities*, *Productivity*, and *Security*, with *Internet Utilities* and *System Utilities* both being included in the first category.

Ending up with 983 applications, we complemented them with 236 free apps to achieve a more balanced distribution. These additional apps were collected by scraping the "Top Ranked Mac App Store Apps" [11] section across all available categories on AppFigures (excluding Games) for the United States region. They were then downloaded using the Mac App Store CLI[12].

Dataset preparation for use in ML classifiers The resulting dataset, containing 1,219 apps, was divided into 2 parts: 80% for training, 20% for testing. Class labels were obtained using data from the AppFigures[13] aggregator collected via web scraping, revealing that our dataset comprised 39.2% free apps, 58.8% freemium apps, and 2% paid apps.

6 Evaluation

6.1 Feature engineering

To train the model, the results of technical and linguistic analysis were first converted into numerical values and assembled into a tabular format, which served as the model input.

The results of the technical analysis were transformed and assigned to separate columns as:

- for each payment framework, we encode its classes and functions with binary values: 0 indicating absence and 1 indicating presence. All framework data were processed consistently using the same approach;
- same for payment endpoints and links to payment pages in their respective quantities;
- same for the receipt file quantifying its in-app purchases records.

The results of the language analysis were processed by generating a separate column for each predefined purchase-related keyword. Each column indicated how many times the corresponding keyword appeared in the text. In addition, separate columns were used for different text sources, i.e. description or privacy policy.

This approach, with assigning the data to different columns, helped the model recognize hidden dependencies between specific frameworks, classes, functions, keywords, and their text source.

[9] iTunes Search API

[10] https://research.macpaw.com/datasets

[11] https://appfigures.com/top-apps/mac-app-store/united-states/top-overall

[12] https://github.com/mas-cli/mas

[13] https://appfigures.com/

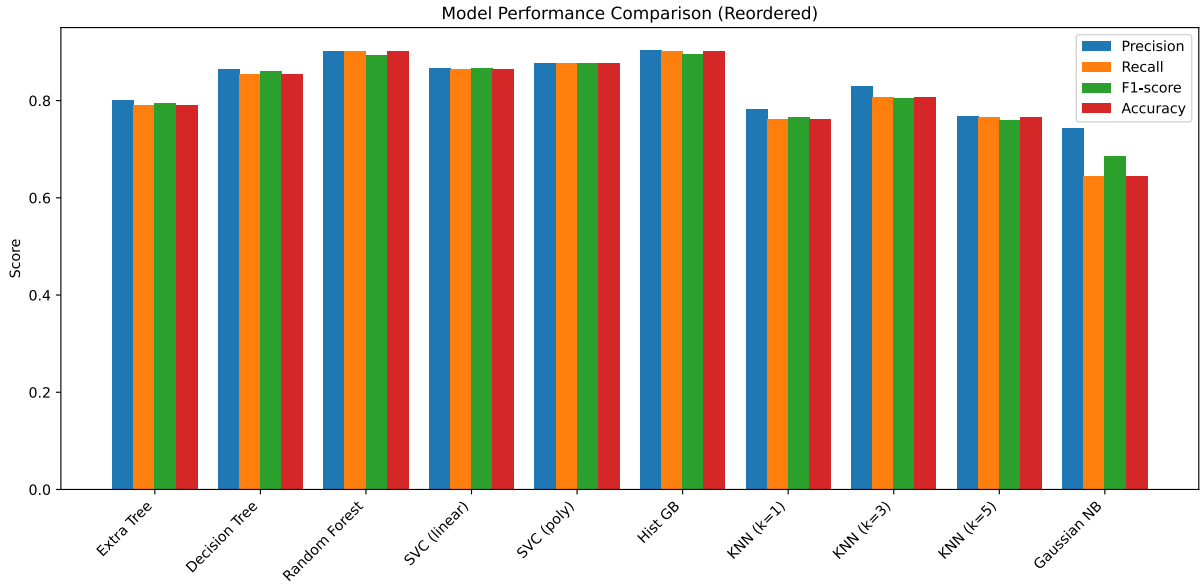

Figure 2: Comparison of Precision, Recall, F1-score, and Accuracy for various models.

6.2 Training and validation

The following supervised machine-learning algorithms were used for predicting an app's monetization class:

- Decision trees (Mienye and Jere, 2024): Decision Tree, Extra Randomized Trees and Random Forest;
- K-Nearest Neighbor (KNN) (Sun et al., 2018): experiments for k=1, 3, 5;
- Naive Bayes (Çınar, 2023): Gaussian Naive Bayes;
- Support Vector Machines (SVM) (Awad and Khanna, 2015): linear and polynomial kernels;
- Gradient Boosting (Chen, 2024): Hist Gradient Boosting Classifier.

The performance of these models was assessed by Precision, Recall, F1 score, and Accuracy metrics, calculated from TP, FP, TN, and FN counts (Vujović et al., 2021). All machine learning models and the evaluation functions used to compute performance metrics were sourced from the Scikit-learn package[14].

The chart in Figure 2 shows that ensemble methods such as Random Forest and Histogram-Based Gradient Boosting outperformed all other classifiers, delivering nearly identical results across the evaluation metrics. Among SVC variants, the polynomial kernel achieved slightly better results across all metrics compared to the linear kernel, owing to its ability to identify nonlinear relationships between features. For K-Nearest Neighbors, increasing k from 1 to 3 improved performance by reducing variance and noise sensitivity, but further increasing k to k=5 degraded results due to oversmoothing and increased bias.

7 Related Work

To the best of our knowledge, there are no publicly available studies whose primary task is the identification of the main monetization model of an app. Therefore, the foundation of this research was laid by the studies in which the major focus was on the classification of software across Android, iOS and macOS operating systems based on their specific characteristics.

7.1 Android

In Sanz et al. (2012), the categorization of Android apps was based on the elements of static analysis, in particular, the analysis of printable strings retrieved from the decompiled Android Package Kit (APK) files along with app permissions. To extend the range of information, the features extracted from the Android Market were used. The authors compared the accuracy of different supervised learning algorithms and found the Bayes Tan classifier the best with 93% accuracy.

[14] https://scikit-learn.org/stable/

Lindorfer et al. (2015) used static analysis for the extraction of meta-information about an app and security-critical Application Program Interface (API) calls, along with dynamic analysis for network operations.

The fact that the potential security vulnerabilities could be injected indirectly into the program by third-party libraries necessitated performing the analysis of their internal functionality. Chen et al. (2016) dedicated the study to detecting the potentially harmful libraries in mobile operating systems using static analysis. The authors claimed that many harmful libraries were the repackaged versions of the legitimate ones, which was a problem similar to the fraudulent apps attempting to hide their monetization.

7.2 iOS

The research related to the analysis of iOS and macOS applications mainly focused on the privacy leaks and malware detection using static and dynamic types of analysis.

In Bhatt et al. (2021), the combination of static and dynamic analysis was used to detect the privacy leaks in iOS apps. Static analysis involved extraction of the used frameworks, classes, and methods from the app's binary with a reverse engineered tool named "otool". The dynamic analysis was used to identify the data accessed by an application at runtime and the network traffic. The model based on the Support Vector Machine (SVM) algorithm was proposed to provide a detection rate of 97%.

In Orikogbo et al. (2016), a significantly larger dataset was used to identify the potential security vulnerabilities in iOS applications compared to the previous research, including more than 43 thousand applications. This paper extended the default static analysis with a new approach —— validating the SSL certificates of domains. To achieve this, the regular expressions were employed to extract all HTTP and HTTPS URLs from applications.

7.3 macOS

For macOS malware detection and classification, a common methodology involved examining the Mach-O binary files and extracting general features, such as the sizes of segments, sections, load commands, symbol tables, the list of dynamic libraries used, and the number of symbols and sections within each segment, etc. (Pajouh et al., 2018; Burgardt, 2022). Bumanglag (2022) and Thaeler et al. (2023) extended these features with data regarding the presence or absence of suspicious strings extracted from the 'c_string' section of the "__TEXT" segment.

7.4 NLP

Natural language processing (NLP) is widely used as a part of different classification algorithms or even as the main approach.

In Zhu et al. (2013), the primary sources of information to perform mobile classification were Web knowledge from the Web search engine, which was combined with contextual features extracted from the real-world context logs. To train the App classifier, MaxEnt, which was successfully applied to a wide range of NLP tasks, was proposed.

Guzman and Maalej (2014) first identified the specific functionality of the application based on users' reviews and then determined the sentiment of the reviews regarding the features of each app. Basic approaches, including noun, verb, and adjective extraction, stopword removal, and lemmatization were used for the text preprocessing. According to Jiang et al. (2019), the information about the app's functionality could also be extracted from its description.

There are also studies that compared the functionality described in the app's description with its actual behavior based on API usage (Gorla et al., 2014). However, there were cases where descriptions were unavailable or significantly changed. This issue was addressed in the article by Shamsujjoha et al. (2021), where a preliminary definition of the application's functionality was proposed based solely on the textual data from the APK file of the application.

8 Conclusions and Future Work

The findings of this research contributed to performing the applications inventory and mapping according to their monetization category. Categorizing apps based on this attribute ensures transparency and informed choices for end users, improved user experience based on avoiding hidden paywalls, and comparing alternatives.

To achieve this goal, an algorithm was developed that combined a set of approaches, including analysis of apps' binary files, resources, collection and processing of descriptions, legal documents, user reviews, and final classification of the apps' dataset.

To validate the applied method, the dataset of

1,219 apps from MAS and MacUpdate was collected. From each app, the aforementioned features were extracted and used to train the classification models, the performance of which was compared using different metrics. The best results were obtained using the Random Forest and Hist Gradient Boosting classifiers, both achieving 0.90 F1-score and 90.1% accuracy.

Despite a quite high rate of classification correctness in this research, its future prospects will involve extending the input features and their analysis. In particular, the application of dynamic analysis to examining the app's runtime behavior, and of unsupervised machine learning algorithms to cluster the extracted features. Additionally, further analysis may include recognizing the relationships between the selected features and the classification validation metrics.

References

Mariette Awad and Rahul Khanna. 2015. Support vector machines for classification. In *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pages 39–66. Springer.

Arpita Jadhav Bhatt, Chetna Gupta, and Sangeeta Mittal. 2021. iabc-al: Active learning-based privacy leaks threat detection for ios applications. *Journal of King Saud University-Computer and Information Sciences*, 33(7):769–786.

Dave Binkley, Marcia Davis, Dawn Lawrie, and Christopher Morrell. 2009. To camelcase or under_score. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 158–167. IEEE.

Kimo Bumanglag. 2022. An application of machine learning to analysis of packed mac malware.

Caio Augusto Pereira Burgardt. 2022. Malware detection in macos using supervised learning.

Hung-Hsuan Chen. 2024. Understanding gradient boosting classifier: Training, prediction, and the role of $gamma\_j$. *arXiv preprint arXiv:2410.05623*.

Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. 2016. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 357–376. IEEE.

Ayşe Çınar. 2023. Multi-class classification with the gaussian naive bayes algorithm. *Journal of Data Applications*, (2):1–13.

Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proceedings of the 36th international conference on software engineering*, pages 1025–1035.

Emitza Guzman and Walid Maalej. 2014. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)*, pages 153–162. Ieee.

He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, David Lo, Xindong Wu, and Zhongxuan Luo. 2019. Recommending new features from mobile app descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4):1–29.

Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. 2015. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In *2015 IEEE 39th annual computer software and applications conference*, volume 2, pages 422–433. IEEE.

Ibomoiye Domor Mienye and Nobert Jere. 2024. A survey of decision trees: Concepts, algorithms, and applications. *IEEE access*.

Damilola Orikogbo, Matthias Büchler, and Manuel Egele. 2016. Crios: Toward large-scale ios application analysis. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 33–42.

Hamed Haddad Pajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo. 2018. Intelligent os x malware threat detection with code inspection. *Journal of Computer Virology and Hacking Techniques*, 14:213–223.

Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. 2004. A rule based approach to word lemmatization. In *Proceedings of IS*, volume 3, pages 83–86. Citeseer.

Eric P Robinson and Yicheng Zhu. 2020. Beyond "i agree": Users' understanding of web site terms of service. *Social media+ society*, 6(1):2056305119897321.

Piyumika Samarasekara, Ridmi Hettiarachchi, et al. 2023. A comparative analysis of static and dynamic code analysis techniques. *Authorea Preprints*.

Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. 2012. On the automatic categorisation of android applications. In *2012 IEEE Consumer communications and networking conference (CCNC)*, pages 149–153. IEEE.

Md Shamsujjoha, John Grundy, Li Li, Hourieh Khalajzadeh, and Qinghua Lu. 2021. Checking app behavior against app descriptions: What if there are no app descriptions? In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 422–432. IEEE.

Jingwen Sun, Weixing Du, and Niancai Shi. 2018. A survey of knn algorithm. *Information Engineering and Applied Computing*, 1(1):1–10.

Andrew Thaeler, Yagmur Yigit, Leandros Maglaras, William J Buchanan, Naghmeh Moradpoor, and Gordon Russell. 2023. Enhancing mac os malware detection through machine learning and mach-o file analysis. In *2023 IEEE 28th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 170–175. IEEE.

Ž Vujović et al. 2021. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications*, 12(6):599–606.

Hengshu Zhu, Enhong Chen, Hui Xiong, Huanhuan Cao, and Jilei Tian. 2013. Mobile app classification with enriched contextual information. *IEEE Transactions on mobile computing*, 13(7):1550–1563.