Analysing User Reviews for Mobile App Security and Privacy

Ujjval Kumar

Uk25@kent.ac.uk

School of Computing

Supervisor: Dr Özgür Kafalı

Abstract

There are over 1.8 million apps available on the IOS Store, and over 3.5 million on the Google Play Store. On average, over 2000 new apps are published daily from both App Stores [1][2], leading to reviews. User reviews can have feature requests, bug reports, security, and privacy issues that allow developers to make sure that the app is being supported properly. It enables users to decide whether to install apps based on other reviews being published, for instance, certain applications might be risky to users' privacy and take their data without consent.

This project aims to collect user reviews from applications in the IOS and Google Play Stores across various categories. which included games, finance, food, shopping, and travel. Different labels will be assigned to the reviews collected, such as bug reports, feature requests, privacy, and security. The gathered information will offer insightful knowledge about user security and privacy.

Keywords

NLP, Mobile applications, User Reviews, Machine Learning, multi label classification, single label classification, Weka, Text Mining.

Introduction

Mobile applications have become a crucial part of our daily lives, with millions of apps available on market stores in this study I used Google Play and IOS store to gather reviews to assess user feedback. Reviews were labelled as feature requests, bug reports, security, and privacy issues. With the number of applications available, there was likely to be issues regarding applications that relates to the labels, which were previously mentioned. Users can rate apps on a 5-star scale to express their opinions and help other users to decide to install, since a lot of information regarding user reviews can be found. Developers need help to keep up with vulnerabilities and issues with their apps due to the vast number of reviews to manage.[3]

User reviews plays an important role in addressing these challenges as it offers important insights into user preferences and concerns about mobile applications such as Information such as feature requests, bug reports, security issues, and privacy issues can all be found in analysing user reviews. By examining these reviews, developers can gain important knowledge about what aspects of their apps need to be improved to make sure they are up-to-date and satisfy users. However, analysing user feedback is a perplexing task, and time-consuming process due to the substantial number of reviews as well as labelling inconsistencies. To help me in this process I used word embeddings, FastText and GloVe (Global Vectors for Word Representation) [10]. To effectively extract meaningful insights.

This report shows the use of machine learning (ML) classification, Natural Language Toolkit's (NLTK) Library, All the different Tools I used to help me with this project as these tools are deemed necessary to use to make this project a success.

Motivation

The motivation for this project came from as there are millions of applications on the IOS and Google Play stores, developers face competition in gaining and maintaining users. Users’

reviews are a valuable source of information as developers are kept up to date on problems that need to be fix.

By classifying user reviews and analysing user feedback into different labels (bug reports, feature requests, security and, privacy issue) Developers can prioritise their workload and concentrate on tasks that require immediate actions. This not only improves the overall quality of their applications but also builds user trust and loyalty.

As there are constant concerns of security, privacy, feature requests, and bug report of the highest importance developers must proactively address any potential vulnerabilities discovered by user reviews, by responding to these vulnerabilities/concerns developers can keep trust and build reputation to attract new users regardless of categories and applications.

Related work

Maalej, W., Kurtanović, Z., Nabil investigated into app review classification techniques, highlighting the importance of text-based methods. It extends previous work by developing a prototype review analytics tool for improved user feedback analysis by developers and researchers. [4]

Donatella Merlini and Martina Rossini Text categorization provides an in-depth explanation and demonstration the use of WEKA software. A tool that implements the most common machine learning algorithms, to perform a Text Mining analysis on a set of documents with examples. [5]

Wajdi Aljedaani, Mohamed Wiem Mkaouer propose an automated classification system for categorizing accessibility user reviews in Android apps based on different accessibility guidelines. It combines manual and machine learning techniques to accurately label reviews, helping developers identify and address accessibility issues promptly for app improvement. [15]

Walid maalehj and Hadeer Nabil propose probabilistic techniques to classify app reviews into bug reports, feature requests, user experiences, and ratings using metadata and natural language processing, aiding app vendors and developers in managing reviews effectively. [19]

Zulfiqar Ali Memon*, Nida Munawar and Maha Kamal focus on extracting feature requests from user reviews for app improvements. It employs a multi-step approach involving feature extraction, topic modelling and sentiment analysis. [20]

The research papers I read throughout the project gave me valuable insights into basic machine-learning techniques relevant to my project. Techniques from these papers, such as word embeddings, preprocessing techniques, and using Weka Software helped me effectively evaluate and refine my model. Overall, these papers helped shape my process and methodology, resulting in the project's success.

Background

Text Classification -

Weka (Waikato Environment for Knowledge Analysis) - Popular software for machine learning tools used for data mining. Developed at the University of Waikato in New Zealand, Weka

provides a comprehensive collection of algorithms for data preprocessing, classification, regression, clustering, association rules mining, and feature selection. [6]

Natural Language Processing (NLP) Techniques -

Natural Language Tool Kit - Leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries. [7]

Methodology

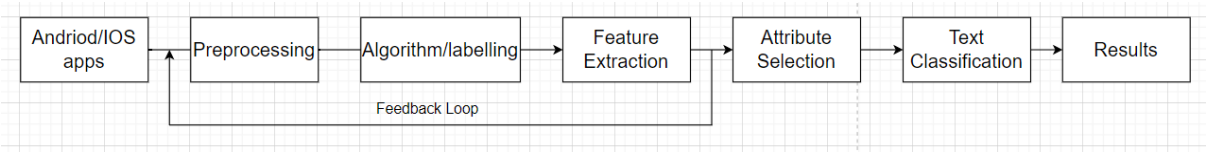

Figure 1 – Proposed methodology for my research.

App Collection

1,187,379 reviews were collected from the Apple App Store and the Google Play Store. These reviews were gathered from a range of categories, which was games, finance, food, shopping, and travel. From each category, four apps were selected, each with a mixture of 1 to 5-star ratings. The number of total reviews scraped is shown in Figures 2 and 3.

Categories	Rating	Reviews
Games		263,652
Brawl Stars	3.6	
Geometry Dash	4.6	
GTA San Andreas	2.8	
Super Mario Bros	3.8	
Finance		229,763
Money Lover Spending	3.9	
Cash-app	1.7	
NatWest	4.8	
Samsung Wallet	2.5	
Food		78,122
Burger King	2.9	
Just Eat	4.6	
Gregg's	4.3	
Subway	3.9	
Shopping		204,124
Amazon	4.1	
Depop	3.6	
Temu	4.7	
Argos	3.4	
Travel		160,262
BA	2.7	
Bookings.com	4.6	
Expedia	4.3	
Hopper	2.9	
Total		935,923

Figure 2 - List of Android Apps which I Scraped.

Categories	Rating	Reviews
Games		43,460
Brawl Stars	4.7	
Geometry Dash	4.5	
GTA San Andreas	3.8	
Super Mario Bros	3.7	
Finance		52,464
Barclays	4.8	
Cash-app	4.8	
NatWest	4.8	
Revolt	4.7	
Food		35,646
Burger King	4.7	
Just Eat	4.7	
Gregg's	4.8	
Subway	4.8	
Shopping		33,070
Amazon	4.8	
Depop	4.8	
Temu	4.7	
Vinted	4.8	
Travel		86,816
BA	4.5	
Bookings.com	4.8	
Expedia	4.8	
Trainline	4.9	
Total		251,456

Figure 3 - List of Apple Apps which I scraped.

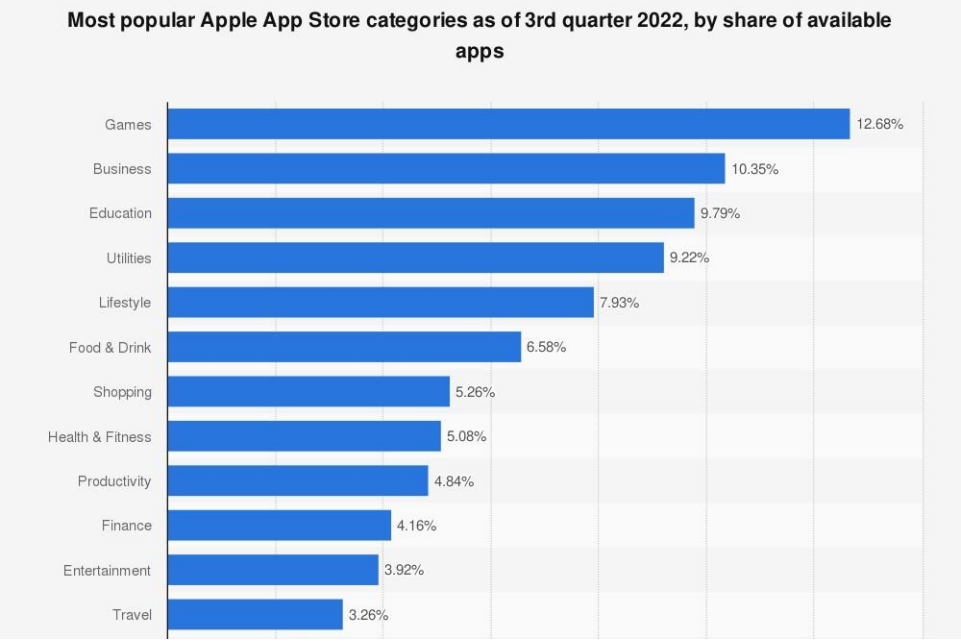

Figure 4 - Most popular Apple App Store categories [8].

The reason for selecting these categories is as follows based of Figure 4:

Games: With 12.68% of the total, they are highly popular and highly viewed in the app store. Given the popularity and size of the gaming community becoming ever more accessible, I assume Android follows a similar pattern.

Food: With 6.58%, of the total, showing the increasing use of applications related to food, restaurant, food delivery, and diet tracking are included in this category.

Shopping: With 5.36%, of the total, this is a growing activity due to the ease of online shopping and its quick delivery time. E-commerce is also becoming increasingly popular, as anything may be bought by users from any location with internet access.

Finance: With 4.16% of the total, this is of the significance of financial transactions conducted through mobile applications. Among other financial activities, finance apps let users invest in stocks, pay bills, and manage their accounts all on their phones/Tablets as of convenience.

Travel apps: With 3.26% of the total, because mobile devices are so widely used for booking and planning trips. Travel apps come with features like hotel and flight bookings, itinerary management, and the destination guides.

Given that the Apple App Store and Google Play Store are the two most popular app stores and have much larger app libraries than other marketplace for example, Amazon App Store. It was decided to use the IOS store and Google Play store as these two marketplaces are well established and well known. Since there are roughly 1,876,058 apps available on the Apple Store as of February 26, 2024 [1], and there are roughly 3,487,155 apps available on the Google Play Store [2]. The Amazon app store, on the other hand, offers roughly 888,698 apps, which is considerably fewer than Apple and Android [9].

Since the Apple App Store and Google Play Store have a wider range of apps for analysing reviews, this allows a wider focus for more in-depth investigation and analysis, which could result in stronger findings.

Using Python's libraries made the process easier to scrape data from the marketplace, as I made multiple web crawler scripts to automatically collect app information, which I used for my dataset. To collect application data from the Apple App Store, I used the app-store-scraper 0.3.5 package, and for Android apps, I used the google-play-scraper 1.2.6 package (Figure 5).

```python
    # Import the required libraries
import csv
from google_play_scraper import Sort, reviews, app

result = app(
    'com.supercell.brawlstars', # app ID
    lang='en',   #language of app review
    country='GB'  #what Region the app is location in
)

is_free = result['price'] == 0 # Check if the app is
    free

result, continuation_token = reviews(
    'com.supercell.brawlstars',
    lang='en',
    country='GB',
    sort=Sort.NEWEST,
    count=10000,
    filter_score_with=5
    # Filter scores equal to  5
)
```

Figure 5: snippet of the Google Play Scraper Code.

Preprocessing

Text cleaning is an essential part of text classification as user reviews are formatted into two parts: the main text and metadata, which are additional, less important details added to the review, for example the review's title, rating, duration, cost. [4]. Which I didn't need as it would have increased noise in my data.

I used Python packages such as pandas, NLTK, re to aid me when creating Python code to clean my data.

These are the Steps I undertook:

1) Remove Special Characters:

Dropping characters that don't contribute to the text's semantic meaning as they are unnecessary information as removing them will reduce noise in the dataset.

2) Replace Emojis:

Converting the meaning of emojis into text since they offer helpful details that may be connected to the user's perception of the apps.

3) Spelling Correction:

Fixing spelling mistake in reviews as users can misspell words rending them useless. By fixing spellings in reviews, accurate analysis will be possible as words will be of proper English.

4) Remove Stop Words:

Stop words are commonly used words that appear frequently in the Text for example “and” "the", "is" but these texts have little semantic value. Removing these words helps to simplify the text data, focusing words that better reflect the review's description.

5) Lowercasing:

Converts all text to lowercase to ensure consistency.

6) Lemmatization:

Lemmatization is the Process of reducing words to their root form, by converting words such as "running" to "run," ensures that similar words are treated equally during text classification.

7) Drop Duplicates:

By dropping duplicated rows according to the 'Review' column, ensures that every review is unique data to the dataset. This enhances the quality of data for analysis and reduces noise.

8) Remove Short Reviews:

By removing reviews with fewer than three words, this reduces the size of the Csv file and removes reviews that are not important and ensure relevant reviews are considered for analysis.

9) Language Detection:

By removing reviews that are not in English, reduces noise and ensures meaningful reviews that are considered for analysis as foreign reviews are redundant information.

10) Tokenization

The goal of this step is to divide the text into individuals Words as tokens, so that each word can be processed separately.

Algorithm

I labelled my Reviews under five different labels, which were Bug report, Feature Request, Security and Privacy.

First, I needed to define what each Label means and Since I created a Keyword List, I needed to decide what words would appear under these labels.

To make my Keyword List I used two methods: First method, was delving into reviews and selecting keywords that could be define under the labels; Second method was using Feature Extraction (word embeddings).

Definitions:

Bug Report: This Term is used to describe situations where users encounter issues that affects the functionality of the application. These issues can vary in severity from minor to

major. for example, in games if the player cannot pick up a gun that is a bug or when a user clicks a button that was meant to do something, but it didn't work. It is important to inform developers of these issues to be fixed.

Feature Request: This term is used to describe where users come up with new ideas to improve an app's functionality, such as improving the user interface or adding new features that could increase the app's rating. This could help developers and users as it improves their experience and satisfaction.

Security: This term is used to describe situations where users find potential vulnerabilities in the application that hackers could exploit, for example code vulnerabilities, flaws in authentications as these are vital information to safeguard users accounts, and developer should always be on the lookout for security threats that appears within their apps.

Privacy: This term refers to issues or violations involving user confidential data. Addressing these concerns is essential to supporting user trust and following the data protection regulations/laws. Examples of these could include unauthorised access to personal information, insufficient data protection measures, or violations of privacy policies like a game app being able to access your photos or camera etc.

The reason I choose these keywords as these terms are widely used in software development because of their clarity and specificity in describing diverse types of feedback or issues, making them effective for categorizing and prioritizing user feedback.

Examples keywords:

Bug Report – 'Unreliable', 'bug', 'fix', 'problem', 'Issue', 'defect', 'crash'.

Feature Request - 'add', 'please', 'could', 'would', 'hope', 'improve', 'miss', 'need', 'suggests', 'wanted'.

Security – 'Security', 'Safety', 'vulnerability', 'Breach', 'Password', 'Permission', 'breached'.

Privacy – 'Privacy', 'Permissions', 'Information', 'Data', 'Location', 'Settings'.

There are words that can come under multiple labels for example Permission, Breached.

Feature Extraction:

Second method I used is Word embeddings to label my reviews, these are semantically aware word representation that allows words with similar meanings to have similar representation in a vector space model [10].

I labelled my data using GloVe and FastText models because, as the research papers emphasise GloVe (Global Vectors for Word Representation) and FastText, as both corpuses have a lot of potential to help improve my labelling which will lead to better and more accurate reviews since it will increase my keyword List .[10][11] Lastly they are the Two most common methods for producing word embeddings.

The paper "Advances in Pre-Training Distributed Word Representation" emphasises the importance of pre-training distributed word representations like GloVe and FastText for different natural language processing tasks, and the importance for different natural

language processing tasks. GloVe captures global co-occurrence statistics of words in a corpus [12].

FastText incorporates sub words information to improve the robustness of word embeddings output. [13] Machine learning techniques can effectively capture semantic similarities between words using the corpus, allowing for more accurate and robust classification of mobile applications based on textual descriptions or reviews.

Using Pre-trained GloVe and FastText models are easily integrated in my models to improve the effectiveness and efficiency of classifying mobile application reviews while optimizing each algorithm. The classification process is more accurate and thorough by utilising the GloVe and FastText word embedding corpus. Resulting in a more thorough understanding and classification of app reviews.

There are Drawbacks of using Word embeddings, as these include potential semantic ambiguity, as models may not fully capture the precise meaning of words, resulting in potential misunderstandings and mislabel reviews. As for example in this case I want Bug to be in context from a technical background not in general bug etc.

To address these issues, it is important to fine-tune parameters, implement approaches to reduce semantic complexities. Furthermore, investigating alternative methods and techniques, such as incorporating other embedding models or LDA, can aid in the performance and reliability of text classification algorithms.

Lastly using these models are computational expensive, the hardware I was using was a Ryzen 7 5700x and 16gb ram it still took me some time to run these models, so it was a balance of finding the best models for this project and which wasn't too expensive and time consuming while having a large enough corpus of words to label my data and to perform classification. In the end I choose FastText model pre-trained word vectors crawl-300d-2M.vec.zip: 2-million-word vectors trained on Common Crawl (600B tokens) and the GloVe model Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download). Overall, striking a balance between the benefits and drawbacks of using these models. Since word embeddings is critical for improving the accuracy and reliability of review labelling [10].

Lastly, there are drawbacks of using a keyword List, first constructing a comprehensive List requires numerous iterations to gather enough words that accurately represent the labels, as this was an iterative process which was time consuming and may not get a perfect list within the constraints of limited time I had.

After several iterations of reviewing the data, the keyword list for each label was refined. The keyword list for Bug contained 39 key words, while Security had 40 key words, Feature Request had 36 key words and Privacy had 30 key words. These lists were carefully constructed to provide a wide range of words that accurately reflect the meaning of each label. However, it is important to note that even with these efforts and the use of word embeddings, reviews may still not be viewed correctly, highlighting the limitations of the methods employed.

Despite these challenges I developed 4 Python scripts to label the reviews into single label and multi label classification algorithms using these methods I talked about previously.

The four algorithm I made which were based on rule-based classification.

1st algorithm is single-label classification with an implicit rule based on keyword counts.

2nd algorithm is also the same as first algorithm but using Fast text model, 20 parameters.

3rd algorithm is rule-based multi-label classification algorithm.

4th algorithm is same as third algorithm but using GloVe embeddings using 5 parameters.

When creating the single label algorithm, as shown in Figure 6, there is a condition that determines which of the following occurs: if the label count, for example, Bug and Feature request, is both one, the algorithm will randomly select which one to review the label; this is done by the random package.

```
    # Lists to store debugging information
    debug_info = []

def classify_review(review):
    review_lower = review.lower()

    label_counts = {'Bug': 0, 'Feature Request': 0, 'Security': 0, 'Privacy': 0}

    for keyword in bug_keywords:
        label_counts['Bug'] += review_lower.count(keyword)

    for keyword in feature_requests:
        label_counts['Feature Request'] += review_lower.count(keyword)

    for keyword in security_keywords:
        label_counts['Security'] += review_lower.count(keyword)

    for keyword in privacy_keywords:
        label_counts['Privacy'] += review_lower.count(keyword)

    max_count = max(label_counts.values())

    debug_info.append({'Review': review, 'Label Counts': label_counts, 'Max Count'
        : max_count})

    # Find all labels with the maximum count
    max_labels = [label for label, count in label_counts.items() if count ==
        max_count]

    if max_count > 0 and max_labels:
        # If there's a tie and labels are present, randomly select one of the tied
            labels
        selected_label = random.choice(max_labels)
        return selected_label
    else:
        # If there's no tie or no labels, return 'Other'
        return 'Other'
```

Figure 6: snippet of single label algorithm.

Lastly with the 2nd algorithm and the 4th algorithm I made PCA Graphs which stands for Principal Component Analysis, is applied to the word embeddings generated by the FastText and GloVe models to visualize how words are clustered based on their semantic similarities. [14]

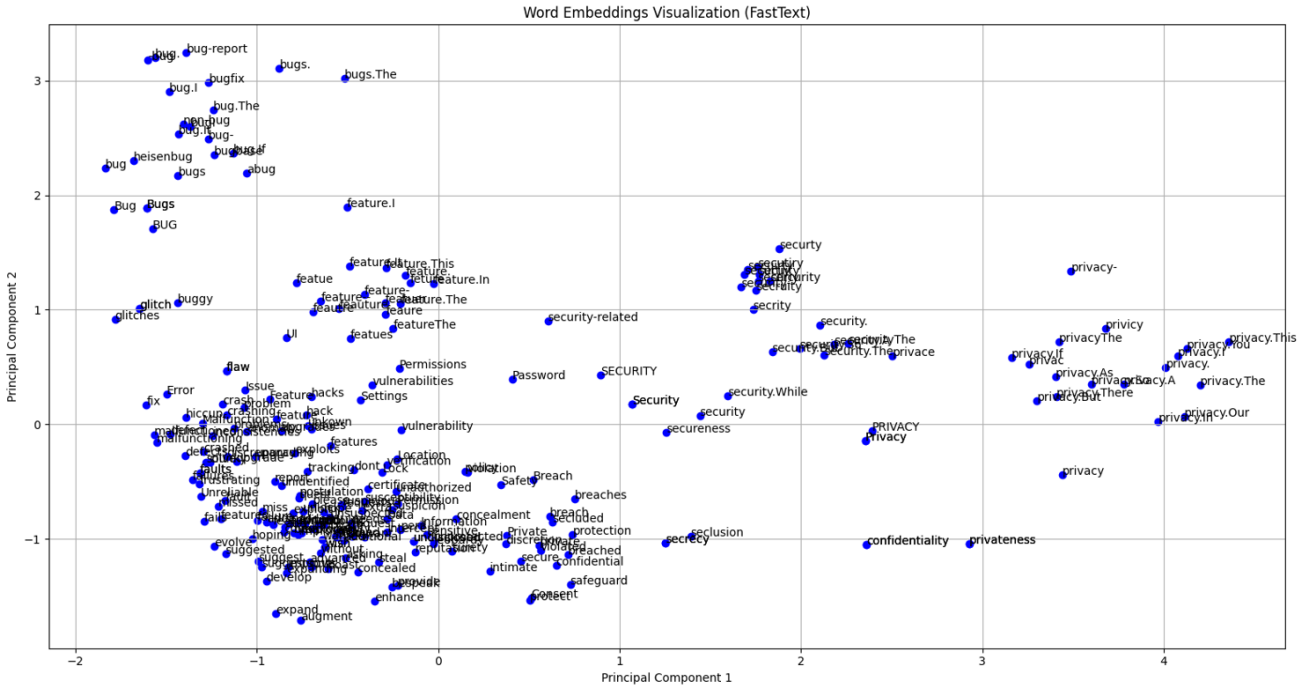

Figure 7: FastText Model shown in a PCA graph.

In Figure 7 uses the FastText model pre-trained word vectors crawl-300d-2M.vec.zip: 2-million-word vectors trained on Common Crawl (600B tokens). I chose this model due to its large and diverse corpus of words, which enables comprehensive coverage of vocabulary. FastText visualisation generated 20 words set as a parameter. Which shows the vector spaces of 20 nearest neighbours to the word ‘bug’, ‘Feature Request’, ‘Security’, ‘Privacy’. Each point on the visualisation represents a word, with words with similar meanings clustered together. For example, words like "defect," "error," and "issue" cluster near "bug," showing semantic similarity and context.

In Figure 8 uses the GloVe model Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip this ensures a large and diverse vocabulary for analysis. Unlike the previous approach, it focused on my predefined list of keywords like 'bug’, 'security’, etc. For each keyword, it retrieves the 5 closest neighbours within the vector space based on semantic similarity. This highlights how the model understands these keywords and their relationships in the context of the corpus it was trained on.

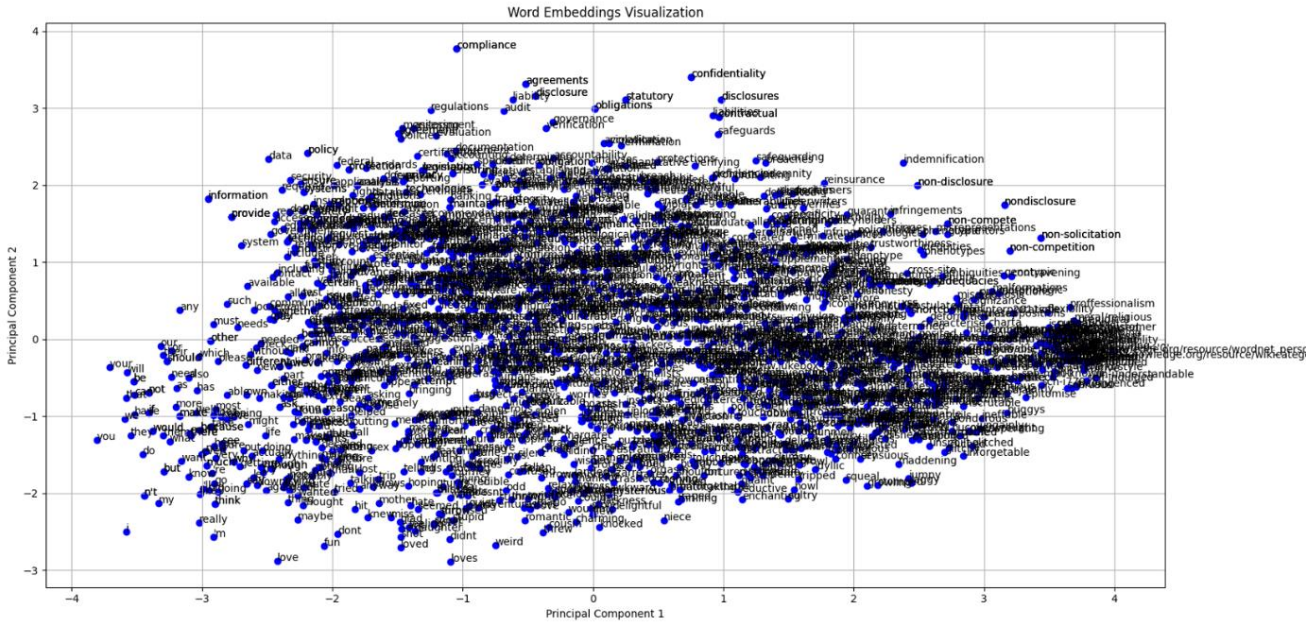

Figure 8: GloVe model shown in a PCA Graph.

Classification –

The dataset was first uploaded to Weka [6] using the ArffViewer tool to convert the csv file into an ARFF file format (Figure 9).

I passed the dataset through the NominalToString filter to handle nominal attributes. Using the common words approach [5]. Then, I used the StringToWordVector(Figure 10) filter to process the dataset to be able to extract features.

File Edit View RFF-Viewer - C:\Users\ujjva\one drive kent\OneDrive - University of Ken

classified_reviews_multi.csv *

Relation: classified_reviews_multi

No.	1: Review Nominal	2: label Nominal
1	clunky ol...	Other
2	quite use...	Feature Request, Bug
3	difficult t...	Privacy, Bug
4	many gli...	Bug, Privacy
5	clunky us...	Other
6	ever give...	Feature Request
7	super ba...	Other
8	fine cam...	Feature Request, Privacy
9	confusin...	Bug
10	app curr...	Feature Request, Bug
11	gate nu...	Feature Request
12	often str...	Privacy
13	app doe...	Bug
14	doesnt al...	Feature Request, Bug
15	you enjo...	Bug, Feature Request
16	new desi...	Feature Request, Privacy
17	slow alw...	Feature Request
18	app getti...	Feature Request, Bug
19	flak syst...	Feature Request
20	intuitive ...	Feature Request
21	intuitive ...	Other
22	cant sav...	Feature Request, Security
23	intuitive l...	Security, Bug
24	work gre...	Bug
25	couldnt f...	Feature Request, Bug
26	didnt fin...	Bug, Feature Request
27	useless ! ...	Security
28	downloa...	Bug, Feature Request
29	arguably...	Bug
30	slow wor...	Privacy, Feature Request
31	away sh...	Feature Request
32	keep fre...	Other

Figure 9: Weka ArffViewer.

In this approach, common words in the dataset were identified and converted into a vector representation matrix, which capture their frequency of occurrence within the dataset. This method enables the extraction of meaningful features from text, allowing for more accurate classification.

Feature extraction process involved modifying parameters using Weka StringtoWordVector settings, which included IDFTransform and TFTransform set as true to perform feature extraction, outputWordCounts set to true. Also, Stemmer and StopwordsHandler were set to NullStemmer and Null. Lastly, the tokenizer was configured as wordTokenizer, with a wordstokeep parameter of 5000. (Figure 10)

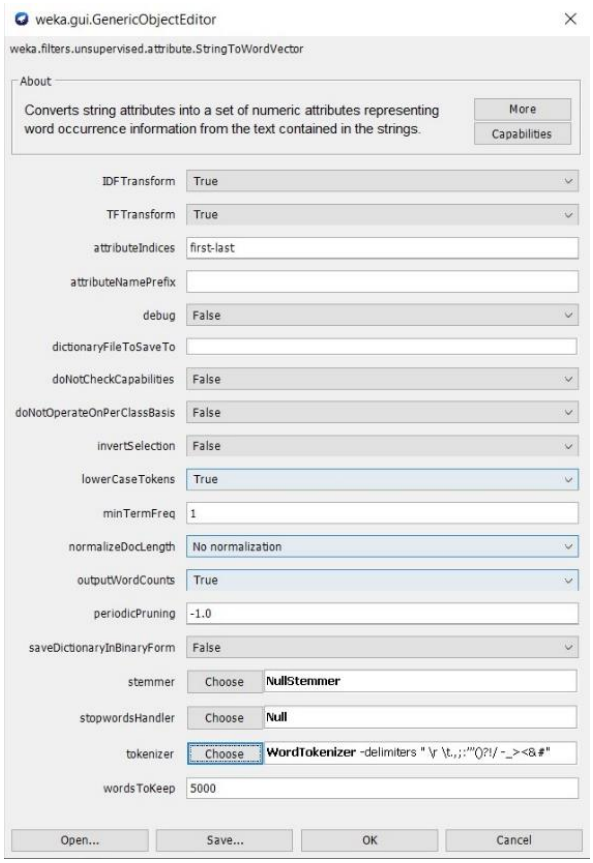

Figure 10: WekaStringtoWord.

The next step was to alter the label attribute as the class attribute. Attribute selection was conducted using the InfoGainAttributeEval method, with the ranker search parameter set to generateRanking - True and a threshold of 0.0 (Figure 11). This approach evaluates the relevance of individual attributes independently of the learning algorithm being used, selecting a subset of attributes based on statistical properties, correlation with the target variable, or other criteria for model training.

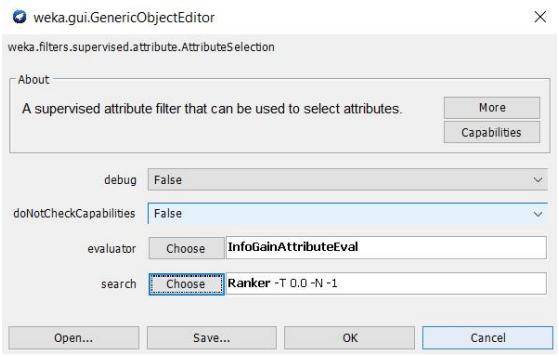

Figure 11 – Weka attribute selection.

After completing the attribute selection process, the next step involved selecting suitable machine learning models for these models.

The classification algorithms I used was Random Forest, Naives bayes Multinomial, J48 Tree, LibSVM C-SVC and Simple Logistic as used in research papers.

Random Forest is an ensemble learning method that constructs multiple decision trees during training. Each tree provides a unique classification, and the algorithm selects the classification with the most votes among all trees to classify new items. This approach enhances accuracy and robustness compared to individual decision trees. [15]

Naives bayes Multinomial is a Bayes theorem-based probabilistic classifier. It works on the basis of term frequency, which considers the frequency with which a word occurs in a document. This model ascertains the probability of a word appearing in a document as well as how frequently it occurs. [16]

J48 Tree comes under decision tree algorithm that organises data hierarchically. Nodes represent feature tests that split the data into subsets, with the root node initiating the tree. Internal nodes divide data based on attributes, leading to final classifications at the leaves. [17]

LibSVM C-SVC (10 cost parameters), Support vector classification (SVM) constructs hyperplanes in a high-dimensional space for classification.[18]

Simple Logistic is a type of logistic regression model that predicts the probability of a binary outcome based on one or more predictor variables. It models the relationship between the binary dependent variable and one or more independent variables by estimating probabilities using a logistic function. [6]

For all these models I did 10-fold cross-validation over percentage split, as this approach creates 10 partitions of the data set. In each partition, 90% of the instances are considered as the training set and 10% as the test set. [10]

TP (True Positives) - parameter determines the predictions labelled correctly by the classifier as positive. [15]

FP (False Positives) - parameter determines the quantity of negative cases incorrectly assumed to be positive via the classifier. [15]

TN (True Negatives) - Parameter determines whether the classifier accurately labels negative predictions. [15]

FN (False Negatives) - Parameter determines the quantity of positive instances that the classifier incorrectly interprets as the negative instances. [15]

Measures	Formula	Definition
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Calculates the closeness of a measured value to the standard value.
Recall	$\frac{TP}{TP+FN}$	Calculates the exact number of positive predictions that are actually observed in the actual class.
Precision	$\frac{TP}{TP+FP}$	Calculates the exact no. of correct predictions out of all the input sample.
F1-score	$\frac{2 \cdot P \cdot R}{P+R}$	Calculates the accuracy from the precision and recall.

Figure 12: Summary of performance measures.[15]

From the metrices, I took the weighted average as part of my results to provide a comprehensive assessment of my model's performance across all classes as discussed in the results section.

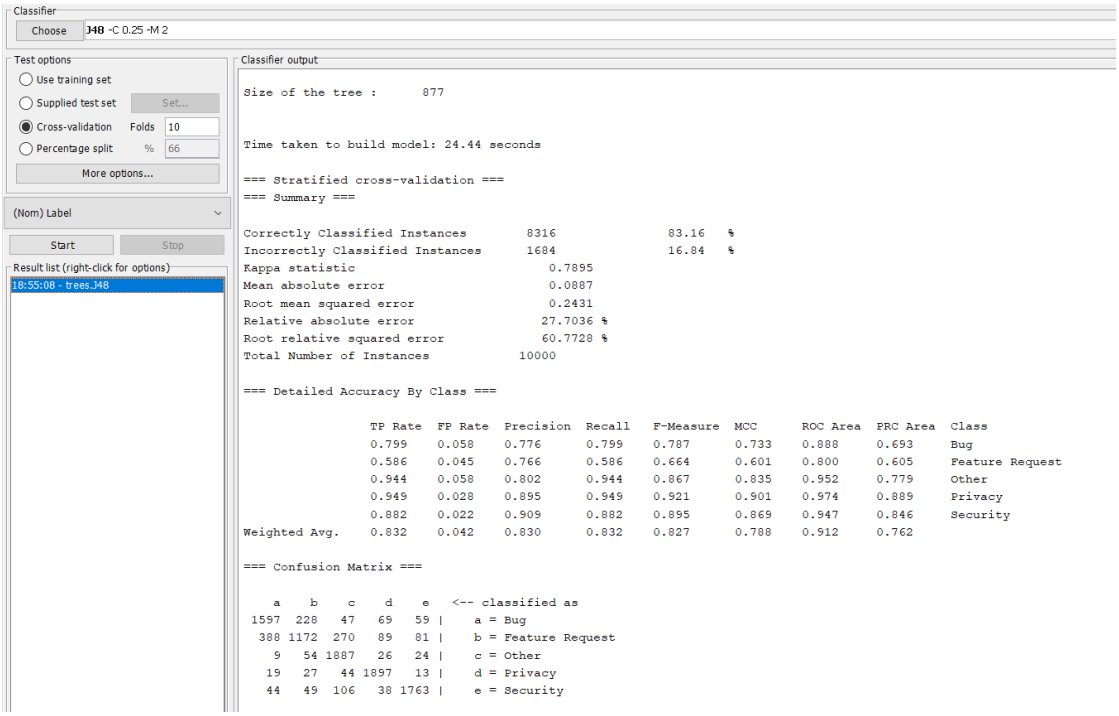

Figure 13: Weka Result Output example.

Results:

The research findings are presented in this section, focusing on the significant comparisons and analyses I conducted to evaluate the performance of my classification models.

1) <u>comparison of 10000 vs 20000 (single label)</u>

when comparing Android 10,000 instances and 20,000 instances datasets. The 10,000 android datasets performed significantly better than the android 20,000 instances, having Higher F-Measure Score. For instance, in the Android single-label distribution with 10,000 instances, the F-measure scores vary from 75.80% to 92.50% (Figure 14) having a 16.70% spread across all algorithms and categories. In contrast, the Android 20,000 single-label

distribution (Figure 15) F-measure scores varies from 72.60% - 92.60% having a 20.00% spread across all algorithms and categories.

Also, when compared to the Apple 20,000 instances dataset (Figure 16), the F-measure scores vary from 74.60% - 91.40%, having a 16.80% difference, they match or exceed the results found in the Android 10,000 single-label dataset. This, could be due to android 10,000 instance data set as less instances means there's no bias and in this dataset all the classes having an equal distribution, meaning that bug label had 2000 instances etc.

Android single labelled 10,000 Instances equally Distributed classes

Category	Algorithm	Performance Metrics (%)		
		Precision	Recall	F-Measure
Games	Naive Bayes Multinomial	77.90	76.80	76.90
	LibSVM C-SVC	92.20	92.20	92.20
	Simple Logistic	92.60	92.60	92.50
	J48	83.20	83.40	82.90
	Random Forest	85.50	85.30	85.00
Finance	Naive Bayes Multinomial	76.20	75.70	75.80
	LibSVM C-SVC	87.40	87.50	87.40
	Simple Logistic	88.90	89.00	88.90
	J48	77.00	77.70	77.10
	Random Forest	80.70	80.90	80.70
Food	Naive Bayes Multinomial	80.80	80.50	80.50
	LibSVM C-SVC	91.60	91.50	91.50
	Simple Logistic	90.90	90.80	90.80
	J48	85.90	85.70	85.60
	Random Forest	89.00	89.00	89.00
Shopping	Naive Bayes Multinomial	78.90	78.60	78.50
	LibSVM C-SVC	90.10	90.00	90.00
	Simple Logistic	90.30	90.30	90.20
	J48	79.80	79.90	79.40
	Random Forest	82.40	82.20	82.00
Travel	Naive Bayes Multinomial	85.00	84.80	84.90
	LibSVM C-SVC	89.40	89.40	89.30
	Simple Logistic	89.30	89.20	89.20
	J48	83.10	83.00	82.90
	Random Forest	85.00	84.80	84.90

Figure 14: Android Classification results.

Android single labelled 20,000 instances Randomly distributed Classes

Category	Algorithm	Performance Metrics (%)		
		Precision	Recall	F-Measure
Games	Naive Bayes Multinomial	77.70	76.00	76.30
	LibSVM C-SVC	89.50	89.50	89.40
	Simple Logistic	90.00	90.10	90.00
	J48	85.40	85.50	85.20
	Random Forest	86.70	86.80	86.70
Finance	Naive Bayes Multinomial	73.50	72.30	72.60
	LibSVM C-SVC	86.00	86.10	86.00
	Simple Logistic	87.10	87.20	87.10
	J48	80.00	80.10	79.90
	Random Forest	82.50	82.70	82.50
Food	Naive Bayes Multinomial	84.10	79.70	81.10
	LibSVM C-SVC	91.90	92.00	91.90
	Simple Logistic	92.60	92.80	92.60
	J48	88.40	88.80	88.40
	Random Forest	87.90	88.40	88.00
Shopping	Naive Bayes Multinomial	78.00	75.30	76.20
	LibSVM C-SVC	88.00	88.20	88.10
	Simple Logistic	89.30	89.50	89.30
	J48	83.40	83.60	83.30
	Random Forest	85.10	85.30	85.20
Travel	Naive Bayes Multinomial	85.60	84.00	84.60
	LibSVM C-SVC	89.20	89.40	89.30
	Simple Logistic	90.70	90.90	90.60
	J48	85.70	86.20	85.80
	Random Forest	85.40	85.60	85.50

Figure 15: Android Classification results.

Apple single labelled 20,000 instances Randomly distributed Classes

Category	Algorithm	Performance Metrics (%)		
		Precision	Recall	F-Measure
Games	Naive Bayes Multinomial	76.70	74.20	74.60
	LibSVM C-SVC	90.70	90.80	90.70
	Simple Logistic	91.40	91.50	91.40
	J48	87.50	87.60	87.50
	Random Forest	89.30	89.40	89.30
Finance	Naive Bayes Multinomial	76.30	75.50	75.70
	LibSVM C-SVC	88.20	88.30	88.20
	Simple Logistic	89.30	89.40	89.30
	J48	83.80	84.10	83.90
	Random Forest	85.20	85.40	85.30
Food	Naive Bayes Multinomial	78.80	76.00	76.90
	LibSVM C-SVC	90.50	90.60	90.50
	Simple Logistic	91.10	91.30	91.10
	J48	86.00	86.30	86.00
	Random Forest	86.10	86.40	86.20
Shopping	Naive Bayes Multinomial	77.40	76.10	76.60
	LibSVM C-SVC	90.00	90.10	90.00
	Simple Logistic	91.10	91.30	91.10
	J48	86.50	86.60	86.50
	Random Forest	86.60	86.80	86.60
Travel	Naive Bayes Multinomial	83.90	82.20	82.80
	LibSVM C-SVC	90.10	90.30	90.10
	Simple Logistic	90.80	90.90	90.80
	J48	84.90	85.20	84.90
	Random Forest	85.80	86.10	85.90

Figure 16: Apple Classification results.

2) comparison of single label vs multi label (overall, all categories/apps)

when comparing dataset with each other Android 10,000 instances, Android 20,000 instances, and Apple 20,000 instances, the single-label datasets continuously showed strong precision, recall, and F-measure scores (Figures 14, 15, and 16). Simple Logistic proved it was the best Classifier in the single label dataset, alongside LibSVM C-SVC, as the best 2 classifiers.

However, with the multi-label datasets (Figures 17 and 18) presented a more complex challenge as multi-label classification makes it possible to assign several labels because reviews usually have numerous labels that more accurately reflect the reviews than single label reviews.

The multi-label datasets for Apple and Android have significantly lower F-measure scores than the single-label datasets, once again, Simple Logistic proved it was the best Classifier in the multi label dataset, alongside LibSVM C-SVC, as the best 2 classifiers, while these two classifiers produce results like the single label. They produce significant higher scores then the rest in the multi label classification models.

When, compared the single-label classification to the multi-label classification usually results in lower classification accuracy, even though it better captures reviews as almost all reviews will contain multiple Issues about the app, as shown by the data. Single-label classification performs better than multi-label classification in terms of recall, precision, and F-measure scores across a variety of datasets and categories.

Android Multi label Classification Randomly distributed Classes

Category	Algorithm	Performance Metrics (%)		
		Precision	Recall	F-Measure
Games	Naive Bayes Multinomial	53.00	53.00	52.90
	LibSVM C-SVC	58.70	58.40	58.10
	Simple Logistic	63.50	63.00	62.50
	J48	50.80	51.80	50.40
	Random Forest	44.00	43.20	43.00
Finance	Naive Bayes Multinomial	58.10	58.50	58.00
	LibSVM C-SVC	64.60	64.50	64.30
	Simple Logistic	66.80	66.90	66.40
	J48	49.90	51.40	50.10
	Random Forest	47.50	47.90	47.10
Food	Naive Bayes Multinomial	56.80	55.60	55.50
	LibSVM C-SVC	62.00	60.40	60.20
	Simple Logistic	63.70	62.00	61.80
	J48	53.50	53.40	52.80
	Random Forest	47.10	46.80	46.60
Shopping	Naive Bayes Multinomial	55.90	55.60	55.50
	LibSVM C-SVC	62.30	62.20	61.70
	Simple Logistic	65.00	64.80	64.30
	J48	51.10	51.90	50.60
	Random Forest	46.20	46.00	45.50
Travel	Naive Bayes Multinomial	55.30	54.60	54.40
	LibSVM C-SVC	59.90	58.40	58.30
	Simple Logistic	61.00	59.60	59.50
	J48	49.30	48.50	48.00
	Random Forest	43.20	42.00	42.10

Figure 17: Android Multi label Classification results.

Apple Multi label Classification Randomly distributed Classes

Category	Algorithm	Performance Metrics (%)		
		Precision	Recall	F-Measure
Games	Naive Bayes Multinomial	55.10	54.50	54.60
	LibSVM C-SVC	62.30	62.40	62.00
	Simple Logistic	64.70	64.40	64.10
	J48	53.80	54.50	53.60
	Random Forest	48.90	49.60	48.70
Finance	Naive Bayes Multinomial	49.50	49.90	49.20
	LibSVM C-SVC	56.90	57.30	56.80
	Simple Logistic	59.90	60.30	59.50
	J48	49.20	50.50	49.40
	Random Forest	44.90	45.60	44.80
Food	Naive Bayes Multinomial	52.10	51.10	51.10
	LibSVM C-SVC	55.00	54.80	54.30
	Simple Logistic	58.20	58.20	57.80
	J48	46.20	47.30	46.10
	Random Forest	40.80	40.90	40.40
Shopping	Naive Bayes Multinomial	62.60	60.70	61.10
	LibSVM C-SVC	71.20	72.60	71.60
	Simple Logistic	72.20	73.20	72.50
	J48	63.00	65.10	63.50
	Random Forest	66.90	68.80	67.70
Travel	Naive Bayes Multinomial	56.00	55.70	55.30
	LibSVM C-SVC	59.10	58.60	58.30
	Simple Logistic	61.50	60.70	60.30
	J48	48.80	49.00	48.10
	Random Forest	43.70	43.00	42.70

Figure 18: Apple Multi label Classification results.

3) comparison of precision between app categories

Comparing precision scores from single-label and multi-label classification datasets across various categories. To work out the Precision of all datasets I took all Precision Result from each algorithm within each category and divided by 5 to get average Precision result.

Android Single label 10,000 (Figure 14) Food Category had the highest precision result 87.64% across all algorithms, followed closely by "Travel", "Games", " Shopping " and, "Finance" categories have relatively lower average precision scores, as the lowest result was Finance with 82.04% the difference between the rest of the categories were roughly 5%.

Android single label 20,00 (Figure 15) shows a similar result as Figure 14 as Food category had the highest Precision 88.98% followed by "Travel," "Games," "Shopping," and "Finance." Which was the lowest with 81.82% difference between the rest of the categories were roughly 7%.

On the Other Hand Apple, single label 20,000 (Figure 16) showed a Different story then compared to Android as Games Category had the Highest precision 87.12%. followed by "Travel", " Shopping ", "Food" and “Finance” have quite similar average precision values roughly around 3% difference, while "Finance" had the lowest average precision was 84.36%.

When compared to Multi label classification the results significantly dropped as for android (Figure 17) "Finance," had the highest Precision 57.38% followed by "Food," "Shopping," "Games," and "Travel." All had similar Precision roughly around 4%. Food had the lowest precision was 53.74%. these results made sense as since this was multi label instead of being 5 classes there was 11 classes.

Lastly when compared to Apple multi label (Figure 18) the precision was all over the place roughly around 17% that was significant as the other Datasets had around 4-7% difference. Shopping Category had the highest Precision 67.18% followed by "Games," "Travel," "Finance," and "Food", which had lowest precision of 50.46%.

In summary Precision Scores varied across different dataset, with some categories consistently performing better than others for example Food, when compared to the multi label classification precision scores significantly dropped as there is added complexity, since there is more instances and more classes that it needs to classify, than with single label.

4) comparison of security/privacy precision vs other classes

There are differences in the results of precision between security, privacy, and other classes over a range of classifiers. In this Analysis, from Figures (19-23) provide a great overview of each Dataset Classes about the Games Category. To do this I took the Precision and Security class of all algorithms and compared them with the average of all the rest of the classes.

Android single label data displayed in (Figure 19), LibSVM C-SVC and SimpleLogistic classifiers continuously demonstrated good precision for both the security, privacy, and other classes. Additionally, the classifiers' precision is constant across all classes, showing that certain classifiers perform better than others and that they are effective at distinguishing security and privacy reviews classes from other classes.

Classifier	Security Class	Privacy Class	Other Class
J48 tree	0.910	0.897	0.784
LibSVM C-SVC	0.939	0.927	0.914
NaiveBayesMultinomial	0.728	0.836	0.776
Random Forest	0.910	0.914	0.817
SimpleLogistic	0.946	0.945	0.912

Figure 19: android 10,000 single label Class Performance Metrics for Different Classifiers.

Android 20,000 instances (figure 20) Simple Logistic stands out for its high precision results across all classes however NaiveBayesMultiomal score was the lowest this shows that this algorithm wasn't optimised or isn't good enough, but the Other Precision classes was higher then both security and Privacy.

Figure 20: android 20,000 single label class Performance Metrics for Different Classifiers.

Classifier	Security Precision	Privacy Precision	Other Precision
Action J48	0.733	0.741	0.859
LibSVM C-SVC	0.795	0.656	0.903
NaiveBayesMultinomial	0.513	0.472	0.799
Random Forest	0.764	0.655	0.875
SimpleLogistic	0.821	0.721	0.905

Lastly Apple 20,000 single label (Figure 21) Once again LibSVM C-SVC and SimpleLogistic stands out showing that in the single label dataset, these classifiers perform well. While With RF classifier varies in performance but with NaiveBayesMultinomial this classifier was the poor in the single label classification, and other precisions scores across all databases except for android 10,000 instances was higher than Security and Privacy accuracy.

Classifier	Security Precision	Privacy Precision	Other Precision
J48	0.658	0.570	0.884
NaiveBayesMultinomial	0.432	0.329	0.786
Random Forest	0.687	0.612	0.902
SimpleLogistic	0.752	0.651	0.920
LibSVM C-SVC	0.713	0.664	0.914

Figure 21: apple 20,000 single label class Performance Metrics for Different Classifiers.

When comparing All of the single label Dataset it is safe to presume that Android 10,000 instances had the best precision score in terms of the performance metrics as shown in Figure 19. When you compare them with the 20,000 instances dataset especially Figure 21 the other Precision class scored higher than Security and Privacy. A reason this could be as the 20,000 instances datasets was all random labels and not uniformed this could lead to bias towards one class.

When I compared Single label to Multi label classification as presumed the Scores for all labels were significantly lower as many factors could affect these scores

In figure 22 LibSVM C-SVC and SimpleLogisitc performs the best.

Classifier	Performance			
	Privacy	Security	Security, Privacy	Other
J48	0.559	0.601	0.580	0.548
NaiveBayesMultinomial	0.527	0.604	0.604	0.519
Random Forest	0.471	0.579	0.622	0.459
SimpleLogistic	0.604	0.707	0.699	0.702
LibSVM C-SVC	0.604	0.696	0.703	0.565

Figure 22: android multi label Class Performance Metrics for Different Classifiers.

Figure 23 shows that NaiveBayesMultinomial and Random Forest classifiers perform worse, when it comes to detecting reviews that are Security & Privacy-related, LibSVM C-SVC and Simple Logistic classifiers outperform them in both the Privacy and Security & Privacy categories.

Classifier	Privacy	Security, Privacy	Other
J48	0.56	0.25	0.537
LibSVM C-SVC	0.655	0.273	0.543
NaiveBayesMultinomial	0.606	0.128	0.605
Random Forest	0.503	0.143	0.607
Simple Logistic	0.657	0.237	0.648

Figure 23: apple multi label Class Performance Metrics for Different Classifiers.

In general, in both Single label and multi label Security and Privacy classes tend to be lower than other classes except from, android 10,000 single label as this is due to having uniform classes. This shows that classifiers may face challenges in accurately identifying and categorizing reviews related to security and privacy concerns, and these models may need to be Finetuned to improve the results.

Overall SimpleLogistic and LibSVM C-SVC classifiers were the best model at classifying privacy and Security classes as many factors comes in play for example single label had 5 classes where with multi label had 11 classes which affected Performance Metrics for Different Classifiers.

Limitations

This study offered valuable insights into the security and privacy implications of Google Play store and IOS app reviews, there are several limitations that I found during this project.

First, with the IOS platform there was an issue in terms of not having the same reviews count for each category as compared to the Android Dataset, Due to the difficulties with the Python-based IOS scraping tool library. The IOS dataset was smaller than expected, which made Comparing Reviews with Android Unfair leading to bias.

Furthermore, the quantity of the keyword list wasn't the best overall, I needed more time to refine the set, and how I incorporated word embeddings models wasn't optimized to the best of its ability, delving more into how the models worked and hyper tuning the algorithm models may improve the classification models' effectiveness and accuracy.

Additionally, there were drawbacks to using Weka for the classification process. When it came to classifying big datasets and running some classification algorithms, my PC couldn’t manage and kept on crashing, as it used too much processing power. Investigating different frameworks, for example, using scikit-learn Library might have helped for more efficient processing and analysis of the dataset, potentially leading to enhanced insights and a classification of larger dataset which could of lead to better analysis.

One major drawback is that the algorithms using words embeddings, are prone to misclassifying reviews, which means that the reviews may not be correctly labelled (Figure 24), could lead to inaccurate analysis. The problem could be solved by refining the

algorithms, which can require a lot of time and resources. Furthermore, the review might not make grammatical sense, as shown in Figure 24.

Figure 24: misclassified Review – Should be Classified as Bug

ǂ	app unresponsive web browser disguised app	Other

Finally, it's essential to recognize that even with my efforts to optimize the models for classification, there a high possibility that the overall performance of classification wasn't to the best, as performing model tuning etc could have had an impact on how successful the classification procedure was.

Conclusion

This study provides important insights for developers as well as users, as the classification results and the results in the corpus can provide useful information to the developers as the corpus results present an understandable pie chart with percentages of reviews labelled as security, privacy, bug, feature request, and other. Since this can show what label needs to be fixed most urgently.

By examining user concerns and preferences regarding security and privacy issues in mobile applications it give users trust about the developers in protecting, listening to their feedback as it is important for everyone involved in this process as it provides insights into user perceptions and expectations.

Future work

Future research could concentrate on improving the algorithms that I used. Fine-tuning the word embeddings models could help improve the model's performance, also the accuracy of the keywords set could have been enhanced by using various feature extraction methods instead of word embeddings, such as Latent Dirichlet Allocation (LDA) [6].

Sentiment analysis and clustering could of have been interesting methods to use, as clustering is used to find common words, sentences, or findings, similar reviews could be grouped together, and this is unsupervised meaning that you don't need to train the data set.

Sentimental analysis is also an interesting approach as, reviews can be grouped as positive, negative, or neutral, which would reveal user preferences and opinions, and you could use Sentimental analysis and Clustering together.

Furthermore, Future research might investigate how long developers responded to user reviews. It could be useful to examine how developers handle user complaints and how well their efforts are to improve their app. For example, if the developers respond quickly to the fix, users will know that they are responsive to user feedback, which may improve their status on the app store.

Lastly, allocating more time and effort into analysing and refining the keyword list that is used in the classification process could significantly enhance the accuracy of the models. As shown in the figures in the corpus, there were a lot of reviews labelled as Other since

keyword selection is an iterative process. Delving deeper into the ‘other’ label reviews to fine-tune the keyword list could lead to better classification results.

References

[1] https://42matters.com/ios-apple-app-store-statistics-and-trends#available-apps-count

[2] https://42matters.com/google-play-statistics-and-trends

[3] Mukherjee, D., Ahmadi, A., Pour, M.V. and Reardon, J., 2020. An Empirical Study on User Reviews Targeting Mobile Apps' Security & Privacy. arXiv preprint arXiv:2010.06371.

[4] Maalej, W., Kurtanović, Z., Nabil, H. and Stanik, C., 2016. On the automatic classification of app reviews. Requirements Engineering, 21, pp.311-331.

[5] Merlini, D. and Rossini, M., 2021. Text categorization with WEKA: A survey. Machine Learning with Applications, 4, p.100033.

[6] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[7] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O’Reilly Media Inc.

[8] https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

[9] https://42matters.com/amazon-appstore-statistics-and-trends

[10] Ebrahimi, F., Tushev, M. and Mahmoud, A., 2021. Classifying mobile applications using word embeddings. ACM Transactions on Software Engineering and Methodology (TOSEM), 31(2), pp.1-30.

[11]Pennington, J., Socher, R. and Manning, C.D., 2014, October. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[12]Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C. and Joulin, A., 2017. Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405.

[13]Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H. and Mikolov, T., 2016. Fasttext. zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.

[14]Tenenbaum, J.B., Silva, V.D. and Langford, J.C., 2000. A global geometric framework for nonlinear dimensionality reduction. science, 290(5500), pp.2319-2323.

[15]Aljedaani, W., Mkaouer, M.W., Ludi, S. and Javed, Y., 2022, March. Automatic classification of accessibility user reviews in android apps. In 2022 7th international conference on data science and machine learning applications (CDMA) (pp. 133-138). IEEE.

[16] Singh, G., Kumar, B., Gaur, L. and Tyagi, A., 2019, April. Comparison between multinomial and Bernoulli naïve Bayes for text classification. In 2019 International conference on automation, computational and technology management (ICACTM) (pp. 593-596). IEEE.

[17] Patra, A. and Singh, D., 2013. A survey report on text classification with different term weighing methods and comparison between classification algorithms. International Journal of Computer Applications, 75(7).

[18] Novakovic, J. and Veljovic, A., 2011, September. C-support vector classification: Selection of kernel and parameters in medical diagnosis. In 2011 IEEE 9th international symposium on intelligent systems and informatics (pp. 465-470). IEEE.

[19]Maalej, W. and Nabil, H., 2015, August. Bug report, feature request, or simply praise? on automatically classifying app reviews. In 2015 IEEE 23rd international requirements engineering conference (RE) (pp. 116-125). IEEE.

[20]Memon, Z.A., Munawar, N. and Kamal, M., 2024. App store mining for feature extraction: analyzing user reviews. Acta Scientiarum. Technology, 46(1).