T.C.
İSTANBUL UNIVERSITY-CERRAHPAŞA
FACULTY OF ENGINEERING

COMPUTER ENGINEERING

DEPARTMENT

INTRODUCTION TO MACHINE LEARNING
FINAL PROJECT

İSMAİL ARDA EVİNÇ
1306200022

2024-2025 FALL

ABSTRACT

This report presents the final project for the Introduction to Machine Learning course, focusing on the classification of user-generated comments from a leading mobile application marketplace. The dataset comprises 229,441 user reviews with ratings categorized into five classes. Three classification algorithms—Logistic Regression, Decision Tree, and k-Nearest Neighbors (k-NN)—were employed to analyze and classify the reviews.

The study emphasizes preprocessing techniques, including handling missing data, text cleaning, and feature transformation using TF-IDF. The dataset was split into training (80%) and test (20%) sets to evaluate model performance. Logistic Regression achieved the highest accuracy of 77.46%, outperforming Decision Tree (71.56%) and k-NN (69.12%). However, all models faced challenges due to class imbalance, with the majority of reviews belonging to the highest rating category.

This project highlights the importance of addressing class imbalance and optimizing feature representation for effective classification. The findings provide valuable insights into the application of machine learning techniques in sentiment analysis, offering a foundation for future research and improvements in user feedback classification systems.

Table of Contents

1. Introduction

This project addresses the problem of classifying user-generated comments from a mobile application marketplace. The primary objective is to analyze user reviews and predict their ratings using machine learning models. By employing Logistic Regression, Decision Tree, and k-Nearest Neighbors (k-NN), the study aims to identify the most effective algorithm for this classification task.

Through comprehensive data preprocessing and feature extraction techniques, the project ensures the dataset is optimized for analysis. The findings provide insights into model performance and highlight challenges such as class imbalance, contributing to advancements in sentiment analysis and user feedback classification.

2. Dataset Overview

The dataset utilized in this project originates from one of the leading mobile application marketplaces, specifically focusing on user-generated comments from a single mobile application. To ensure high-quality data and a sufficient volume of user feedback, the dataset was collected from the top-ranked paid mobile game on the platform. This choice was made to minimize noise and maximize the relevance of the data, given that top-performing paid applications often receive a greater number of detailed and constructive user reviews. By leveraging data from this specific source, the project benefits from insights derived from a robust and representative sample of user opinions, which enhances the reliability and applicability of the findings.

Dataset Description:

- **Number of Examples (Rows):** 229,441
- **Number of Classes:** 5 (representing ratings)

3. Methodology

The methodology of this project involves a systematic approach to preparing, analyzing, and modeling the dataset to draw meaningful insights. The steps taken are outlined below:

3.1 Data Preprocessing

Before conducting any analysis, the dataset underwent a thorough preprocessing phase to ensure its quality and suitability for modeling. This involved:

- **Handling Missing Data:** Rows with incomplete or null values were identified and either imputed using suitable methods or removed to maintain data integrity.
- **Data Cleaning:** Text-based user reviews were cleaned by removing unnecessary symbols, special characters, and stopwords. Additionally, duplicate entries were removed.
- **Data Transformation:** User ratings were encoded into categorical variables to facilitate classification tasks. Text data was vectorized using techniques like TF-IDF to convert it into numerical form suitable for machine learning algorithms.

3.2 Training and Test Set Split

The dataset was split into two subsets:

- **Training Set:** 80% of the data was used to train the classification models.
- **Test Set:** 20% of the data was reserved to evaluate model performance.

This split ensured that the models were trained on a majority of the data while leaving enough examples to test their generalizability.

3.3 Classification Models

To analyze the dataset and classify user reviews, three classification algorithms were employed:

- **Decision Tree Classifier:**
  A tree-based model was used to classify the reviews by learning decision rules derived from the features in the dataset. Decision Trees are interpretable and effectively handle nonlinear relationships.
- **k-Nearest Neighbors (k-NN):**
  This algorithm classified reviews based on their similarity to existing data points. It evaluates the "distance" between feature vectors and assigns the majority class of the nearest neighbors to a new data point.

- **Logistic Regression:**
  A linear model was applied to predict the likelihood of a user rating belonging to a particular class. Logistic Regression is computationally efficient and provides probabilities for classification, making it a robust baseline method.

3.4 Model Evaluation

The performance of the classification models was evaluated using key metrics:

- **Accuracy:** The proportion of correct predictions among the total number of predictions.
- **Precision:** The ratio of true positive predictions to the total positive predictions for each class.
- **Recall:** The ratio of true positive predictions to the actual positive instances for each class.
- **F1-Score:** The harmonic mean of precision and recall, balancing the trade-off between the two metrics.

These metrics were calculated for all models, and their results were compared to identify the best-performing algorithm. The results provided insights into the strengths and weaknesses of each model in classifying user ratings.

4. Results

The results section provides an overview of the performance of the classification models applied to the dataset. Each model's accuracy and detailed classification metrics (precision, recall, and F1-score) are presented to evaluate their effectiveness in predicting user ratings.

4.1 Decision Tree Classifier

The Decision Tree classifier achieved an overall accuracy of **71.56%**, demonstrating a reasonable ability to classify user ratings. Below are the detailed metrics for each class:

Class	Precision	Recall	F1-Score	Support
1	0.54	0.58	0.56	29,470
2	0.18	0.13	0.15	9,105
3	0.16	0.11	0.13	11,548
4	0.18	0.10	0.13	22,840
5	0.83	0.91	0.87	155,478

Overall Accuracy: 71.56%

The Decision Tree classifier performed well on class 5 (highest-rated reviews) but struggled with lower-rated reviews (classes 2, 3, and 4), which were harder to distinguish due to their overlapping characteristics.

4.2 k-Nearest Neighbors (k-NN)

The k-NN classifier achieved an accuracy of **69.12%**, slightly lower than the Decision Tree. The metrics are summarized below:

Class	Precision	Recall	F1-Score	Support
1	0.53	0.27	0.36	29,47
2	0.17	0.02	0.03	9,105
3	0.18	0.05	0.07	11,548
4	0.20	0.06	0.10	22,84
5	0.73	0.95	0.83	155,478

Overall Accuracy: 69.12%

While k-NN effectively classified the most common class (5), it faced significant challenges with less frequent classes, especially class 2, due to its sensitivity to class imbalance and outliers.

4.3 Logistic Regression

Logistic Regression outperformed the other models, achieving an accuracy of **77.46%**. The detailed metrics are provided below:

Class	Precision	Recall	F1-Score	Support
1	0.60	0.81	0.69	29,470
2	0.24	0.04	0.07	9,105
3	0.29	0.09	0.14	11,548
4	0.33	0.05	0.09	22,840
5	0.84	0.97	0.90	155,478

Overall Accuracy: 77.46%

Logistic Regression demonstrated superior performance across most classes, particularly in accurately identifying the most frequent class (5). Its ability to generalize better than the Decision Tree and k-NN classifiers suggests that it is well-suited for the dataset's characteristics.

4.4 Comparative Analysis

The models' performance is summarized as follows:

Model	Accuracy
Decision Tree	71.56%
k-Nearest Neighbors (k-NN)	69.12%
Logistic Regression	77.46%

Key Observations:

- Logistic Regression emerged as the best-performing model in terms of accuracy and F1-score, indicating its robustness for this dataset.

- Both Decision Tree and k-NN struggled with lower-rated classes, possibly due to the inherent class imbalance in the dataset.
- The dataset's heavy skew toward class 5 highlights the need for strategies such as oversampling, undersampling, or weighted loss functions to improve classification performance for underrepresented classes.

This analysis provides a comprehensive understanding of the models' strengths and weaknesses, offering a foundation for future improvements.

5. Discussion

The results of this study highlight several important aspects of the dataset and the classification models applied. The discussion focuses on the performance, strengths, limitations, and potential improvements for each model.

5.1 Performance Analysis

Logistic Regression emerged as the best-performing model, achieving the highest accuracy of **77.46%**. This result can be attributed to its ability to generalize well across the dataset, particularly in identifying the dominant class (rating 5). In contrast, the Decision Tree and k-Nearest Neighbors (k-NN) classifiers achieved lower accuracies of **71.56%** and **69.12%**, respectively.

The dominance of class 5 (representing the highest rating) heavily influenced the models' performance. With approximately **67.8%** of the data belonging to this class, the models tended to favor this majority class, leading to high precision and recall for class 5. However, the less frequent classes (ratings 2, 3, and 4) were significantly harder to classify, as evidenced by their low F1-scores.

5.2 Challenges Encountered

One of the primary challenges was the **class imbalance** in the dataset, with a majority of the reviews falling into the highest rating category. This imbalance skewed the model predictions toward the dominant class, reducing the ability to accurately classify lower ratings.

Additionally, the overlap in features among the minority classes contributed to the difficulty in distinguishing between them. For instance, ratings of 2, 3, and 4 may share similar textual features, making them harder to separate.

5.3 Implications of Findings

The findings from this study provide valuable insights into the effectiveness of classification models for user-generated comments in mobile applications. While Logistic Regression performed well, the limitations encountered highlight the importance of addressing class imbalance and improving feature representation for a more accurate and robust classification system.

By implementing the suggested improvements, future studies could achieve better performance across all classes, ultimately contributing to a deeper understanding of user feedback and enhancing decision-making processes for mobile app developers.

6. Conclusion

This project aimed to explore the application of classification models on a dataset of user-generated comments from a top-ranked paid mobile game. By implementing three different classification algorithms—Logistic Regression, Decision Tree, and k-Nearest Neighbors (k-NN)—we were able to assess their effectiveness in predicting user ratings from textual data.

The results indicated that Logistic Regression outperformed the other two models, achieving the highest accuracy of **77.46%**. While this model showed a solid ability to predict the majority class (rating 5), it struggled with accurately predicting minority classes, highlighting the challenge of class imbalance in the dataset. The Decision Tree model achieved a moderate accuracy of **71.56%**, and the k-NN model performed the least well with an accuracy of **69.12%**.

Despite the promising results, the performance of all models was heavily influenced by the class imbalance, with a large proportion of reviews belonging to the highest rating category. This imbalance, along with challenges in feature differentiation between minority classes, limited the effectiveness of all models in accurately classifying lower ratings.

To improve future performance, techniques such as addressing class imbalance through oversampling, using more advanced feature extraction methods, and experimenting with more complex models like Random Forest or Neural Networks could be applied. Additionally, further research could focus on optimizing hyperparameters and exploring additional features to better capture the complexities of user feedback.

In conclusion, while the models applied in this study showed varying levels of success, they also pointed to areas for improvement in classification accuracy. The findings contribute to the broader field of sentiment analysis and user feedback interpretation, providing a foundation for future work in this domain.

7. References

- The dataset used in this project was prepared based on the application available at the following link: "https://play.google.com/store/apps/details?id=com.scopely.monopolygo&hl=en", and the data was collected in October 2023.

8. Appendix

You can find all the metarials used for this project in the repository on following link:

https://github.com/ArdaEVN/IML

Scraper For Dataset Preperation:

```python
import sys
import csv
from google_play_scraper import Sort, reviews_all

if len(sys.argv) > 1:
    app_id = sys.argv[1]  # Get app ID from the first command-line argument
else:
    app_id = "io.supercent.pizzaidle&hl=en&gl=US"  # Default app ID

all_results = []
for x in range(1, 6):
    results = reviews_all(

        app_id,
        sleep_milliseconds=2,  # defaults to 0
        lang='en',  # defaults to 'en'
        country='us',  # defaults to 'us'
        sort=Sort.NEWEST,  # defaults to Sort.MOST_RELEVANT
        filter_score_with=x  # defaults to None(means all score)
    )
    all_results.extend(results)

```

```
23. if __name__ == '__main__':
24.     f = open('{}_reviews.csv'.format(app_id), 'w', encoding="utf8")
25.     output = csv.writer(f)
26.     output.writerow(all_results[0].keys())  # header row
27.
28.     for row in all_results:
29.         output.writerow(row.values())
30.     f.close()
31.
```

Preprocessing Code For The Dataset:

```
 1. import sys
 2. import pandas as pd
 3. import re
 4. import nltk
 5. from nltk.corpus import stopwords
 6. from nltk.tokenize import word_tokenize
 7. from nltk.stem import WordNetLemmatizer
 8. from bs4 import BeautifulSoup
 9.
10. # Download necessary NLTK datasets
11. nltk.download('stopwords')
12. nltk.download('punkt')
13. nltk.download('punkt_tab')
14. nltk.download('wordnet')
15. nltk.download('omw-1.4')
16.
17. def preprocess_text(text):
18.     try:
19.         # Initialize the lemmatizer
20.         lemmatizer = WordNetLemmatizer()
21.
22.         # Convert to lowercase
23.         text = text.lower()
24.         # Remove URLs
25.         text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
26.         # Remove HTML tags
27.         text = BeautifulSoup(text, "html.parser").text
28.         # Remove punctuation and special characters
29.         text = re.sub(r'[^\w\s]', '', text)
30.         # Tokenize the text
31.         word_tokens = word_tokenize(text)
32.         # Load stopwords
33.         stop_words = set(stopwords.words('english'))
34.         # Remove stopwords and lemmatize the words
35.         filtered_text = [lemmatizer.lemmatize(word) for word in word_tokens if word not in
stop_words]
36.         # Join words back to string
37.         return ' '.join(filtered_text)
38.     except Exception as e:
39.         print(f"Error during text preprocessing: {e}")
40.         return ''
41.
42.
43. if __name__ == "__main__":
44.     input_filename = "230000_monopoly.csv"  # Use your dataset's name directly
45.     output_filename = "updated_reviews_230000_monopoly.csv"  # Output file name
46.
47.     stop_words = set(stopwords.words('english'))
48.     lemmatizer = WordNetLemmatizer()
```

```
49.
50.     try:
51.         df = pd.read_csv(input_filename)
52.         # Assuming the reviews are in the 'content' column
53.         df['processed_reviews'] = df['content'].fillna('').apply(preprocess_text)
54.         df.to_csv(output_filename, index=False)
55.         print(f"Processed reviews saved to {output_filename}")
56.     except FileNotFoundError:
57.         print(f"File not found: {input_filename}")
58.     except pd.errors.EmptyDataError:
59.         print(f"No data: {input_filename} is empty.")
60.     except Exception as e:
61.         print(f"An error occurred: {e}")
62.
```

Training Logistic Regression Model:

```
 1. import pandas as pd
 2. from sklearn.model_selection import train_test_split
 3. from sklearn.feature_extraction.text import TfidfVectorizer
 4. from sklearn.linear_model import LogisticRegression
 5. from sklearn.metrics import accuracy_score, classification_report
 6. from sklearn.pipeline import make_pipeline
 7.
 8. def preprocess_text(text):
 9.     # Assuming text is already preprocessed in the 'processed_reviews' column
10.     return text
11.
12. # Load dataset
13. df = pd.read_csv("updated_reviews_230000_monopoly.csv")
14.
15. # Preprocess the reviews (if needed, we are assuming it's already done)
16. df['processed_reviews'] = df['processed_reviews'].fillna('')
17.
18. # Split data into training and testing sets
19. X = df['processed_reviews']
20. y = df['score']  # assuming 'score' is the label you want to predict
21. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
22.
23. # Create a Logistic Regression classifier pipeline
24. pipeline = make_pipeline(TfidfVectorizer(stop_words='english'),
LogisticRegression(max_iter=1000))
25.
26. # Train the model
27. pipeline.fit(X_train, y_train)
28.
29. # Predict on the test set
30. y_pred = pipeline.predict(X_test)
31.
32. # Evaluate the model
33. accuracy = accuracy_score(y_test, y_pred)
34. print(f"Logistic Regression Model Accuracy: {accuracy * 100:.2f}%")
35. print(classification_report(y_test, y_pred))
36.
37. # Save the trained model (optional)
38. import joblib
39. joblib.dump(pipeline, 'logistic_regression_model.pkl')
40.
```

Training Decision Tree Model:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import make_pipeline

def preprocess_text(text):
    # Assuming text is already preprocessed in the 'processed_reviews' column
    return text

# Load dataset
df = pd.read_csv("updated_reviews_230000_monopoly.csv")

# Preprocess the reviews (if needed, we are assuming it's already done)
df['processed_reviews'] = df['processed_reviews'].fillna('')

# Split data into training and testing sets
X = df['processed_reviews']
y = df['score']  # assuming 'score' is the label you want to predict
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Decision Tree classifier pipeline
pipeline = make_pipeline(TfidfVectorizer(stop_words='english'),
DecisionTreeClassifier(random_state=42))

# Train the model
pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Model Accuracy: {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred))

# Save the trained model (optional)
import joblib
joblib.dump(pipeline, 'decision_tree_model.pkl')

```

Training KNN Model:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import make_pipeline

def preprocess_text(text):
    # Assuming text is already preprocessed in the 'processed_reviews' column
    return text
```

```
11.
12. # Load dataset
13. df = pd.read_csv("updated_reviews_230000_monopoly.csv")
14.
15. # Preprocess the reviews (if needed, we are assuming it's already done)
16. df['processed_reviews'] = df['processed_reviews'].fillna('')
17.
18. # Split data into training and testing sets
19. X = df['processed_reviews']
20. y = df['score']  # assuming 'score' is the label you want to predict
21. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
22.
23. # Create a k-Nearest Neighbors classifier pipeline
24. pipeline = make_pipeline(TfidfVectorizer(stop_words='english'), KNeighborsClassifier())
25.
26. # Train the model
27. pipeline.fit(X_train, y_train)
28.
29. # Predict on the test set
30. y_pred = pipeline.predict(X_test)
31.
32. # Evaluate the model
33. accuracy = accuracy_score(y_test, y_pred)
34. print(f"k-Nearest Neighbors Model Accuracy: {accuracy * 100:.2f}%")
35. print(classification_report(y_test, y_pred))
36.
37. # Save the trained model (optional)
38. import joblib
39. joblib.dump(pipeline, 'knn_model.pkl')
40.
```