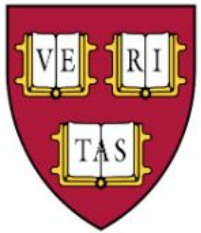
Final Project

E-Cycling Through Insights: NLP-Based Topic Modeling and Classification of E-Bike App Reviews

Hannah Halvorsen



CSCI S-89B Introduction to Natural Language Processing
Fall 2024
Harvard Extension School

Problem Statement and Motivation

Problem Statement:
In the e-bike industry, multiple companies provide smartphone applications to enhance the user experience, offering features like ride tracking, battery management, and connectivity with wearable devices. However, understanding user feedback from app store reviews can be challenging due to the volume and variability of content. The goal of this project is to apply Natural Language Processing (NLP) techniques to automatically identify key themes and categories from user reviews of various e-bike apps. I aim to discover common issues (e.g., connectivity problems, usability concerns) and assess how each company's app performance aligns with user expectations.

Motivation and Goals:
By uncovering underlying topics through LDA-based topic modeling and then classifying new reviews into these categories using a supervised model built with Keras/TensorFlow, I hope to provide insights into user satisfaction, highlight areas for improvement, and guide future app development efforts.

Data Sources (Google Play, Apple App Store)

I collected English-language user reviews from both the Google Play Store and the Apple App Store for several leading e-bike manufacturers' apps: Specialized, Trek, Giant, Canyon, and Cannondale. I used their public APIs or scraping libraries to gather thousands of reviews. After filtering by language (English) and cleaning the text, I ended up with a combined dataset of approximately 2,000+ reviews.

```python
from google_play_scraper import reviews, Sort
import pandas as pd

# Function to scrape reviews from the google playstore
def scrape_playstore_reviews(app_id, max_reviews=1000):
    result, _ = reviews(
        app_id,
        lang="en",        # Language: English
        sort=Sort.NEWEST,
        count=max_reviews
    )
    return pd.DataFrame(result)

# Scrape reviews for each app
app_id = "com.specialized.android"
reviews_specialized = scrape_playstore_reviews(app_id, max_reviews=5000)

app_id = "com.trekbikes.trekcentral"
reviews_trek = scrape_playstore_reviews(app_id, max_reviews=5000)

app_id = "com.GiantGroup.app.RideControl2"
reviews_giant = scrape_playstore_reviews(app_id, max_reviews=5000)

app_id = "com.canyon.connected"
reviews_canyon = scrape_playstore_reviews(app_id, max_reviews=5000)

app_id = "com.cannondale.app"
reviews_cannondale = scrape_playstore_reviews(app_id, max_reviews=5000)
```

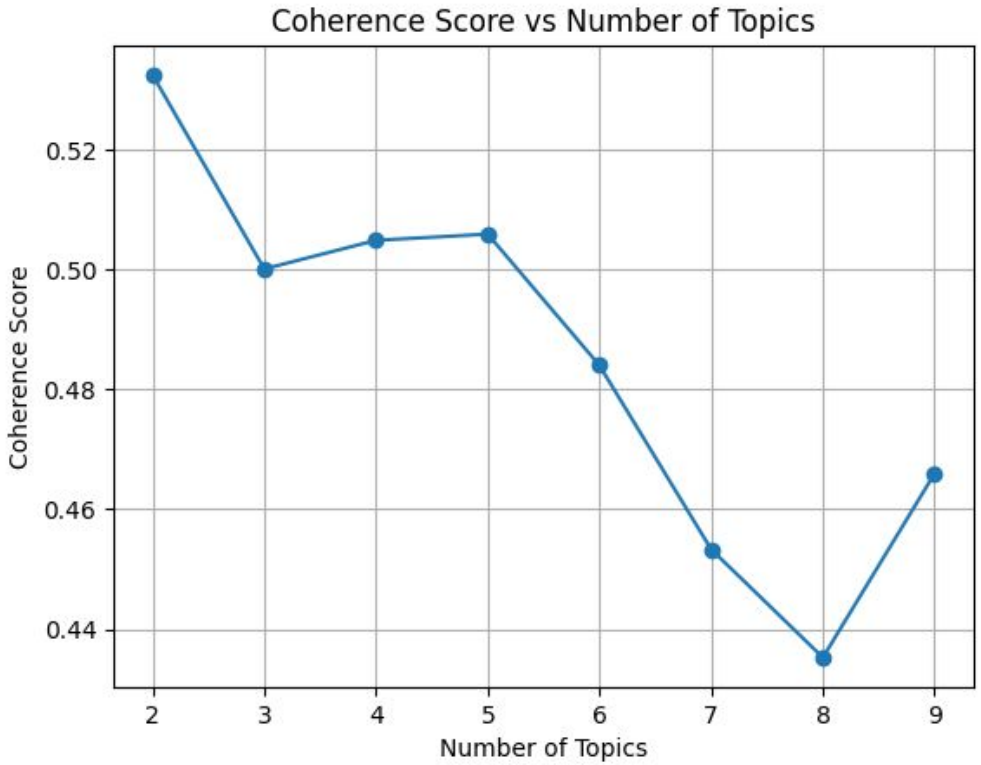
Preprocessing Steps and NLP Pipeline (Text Cleaning, Tokenization)

After scraping, I performed tokenization, lemmatization, and stopword removal. I removed company names and common irrelevant terms to better expose meaningful topics. I also kept certain stopwords in that I felt had sentiment value such as “very” or “most”. I then examined the most common tokens and verified data quality. This step ensures the subsequent LDA and classification models receive cleaner input.

```python
# Tokenization and lemmatization function with emoji support
def preprocess_text(text):
    if not isinstance(text, str):  # Handle non-string entries gracefully
        return []
    tokens = word_tokenize(text)  # Tokenize to preserve emojis and punctuation
    processed_tokens = []
    for token in tokens:
        if emoji.is_emoji(token):  # Keep emojis as-is
            processed_tokens.append(token)
        elif token.lower() == "as":  # Preserve "as"
            processed_tokens.append("as")
        elif token.lower() == "has":  # Preserve "as"
            processed_tokens.append("has")
        elif token.lower() == "was":  # Preserve "as"
            processed_tokens.append("does")
        elif token.lower() == "does":  # Preserve "as"
            processed_tokens.append("was")
        elif token.lower() in ["dont", "don't"]:  # Standardize "don't" to "dont"
            processed_tokens.append("dont")
        elif "/" in token:  # Keep words with slashes
            processed_tokens.append(token)
        elif "'" in token:  # Remove apostrophe but keep the word
            token = token.replace("'", "")
            processed_tokens.append(token)
        elif token.isalnum():  # Retain alphanumeric tokens
            processed_tokens.append(lemmatizer.lemmatize(token.lower()))
        elif any(char.isalnum() for char in token):  # Retain symbols with alphanumerics
            cleaned_token = ''.join(char for char in token if char.isalnum() or emoji.is_emoji(char))
            processed_tokens.append(cleaned_token)
    return processed_tokens
```

Topic Modeling with LDA

I applied LDA to identify underlying themes. I experimented with various numbers of topics and settled on 5 topics based on coherence scores. I labeled these topics to create human-readable categories, e.g., "Battery, Mapping, and General Updates" or "Data, Sensor, and Connectivity Issues."



Final Topic Categories (List Topics and Keywords)

```
topic_labels = {
    0: "Battery, Mapping, and General Updates",
    1: "Usability, Screen Features, and Speed",
    2: "Data, Sensor, and Connectivity Issues",
    3: "Account Management and Login Issues",
    4: "Features, Interface, and Usability"
}
```

```
feature_names = vectorizer.get_feature_names_out()
for idx, topic in enumerate(lda_model.components_):
    print(f"Topic {idx}:")
    print([feature_names[i] for i in topic.argsort()[-15:]])
```

```
Topic 0:
['ca_nt', 'feature', 'setting', 'battery', 'great', 'data', 'update', 'map', 'so',
Topic 1:
['speed', 'all', 'now', 'so', 'update', 'show', 'map', 'time', 'as', 'screen', 'ne
Topic 2:
['would_be', 'data', 'no', 'sensor', 'so', 'as', 'now', 'all', 'use', 'great', 'ti
Topic 3:
['was_not', 'create_an', 'update', 'so', 'need', 'can_t', 'log_in', 'use', 'even',
Topic 4:
['doesn_t', 'new', 'as', 'now', 'see', 'one', 'like', 'feature', 'need', 'connect'
```

Building a Classification Model with Keras

Architecture:
The baseline model was a sequential Keras model consisting of:

a. An `Embedding` layer with random initialization,
b. A Bidirectional LSTM layer (with 64 units),
c. A Dense layer for classification, with `softmax` output for multiple classes.

Data Preparation:
Reviews were tokenized and converted to integer sequences using a Keras `Tokenizer`. All sequences were padded to a fixed length (e.g., 100 tokens) to handle variable-length reviews.

Training Details:

d. **Loss Function:** Categorical crossentropy, suitable for multi-class classification.
e. **Optimizer:** Adam with a default learning rate.
f. **Batch Size & Epochs:** Typically batch sizes of 32 and around 10-15 epochs. Early stopping was considered if validation loss failed to improve.
g. **Validation Split:** A portion of the training set (e.g., 20%) was used for validation during training.

```
X_text = all_reviews['processed_text'].tolist()
y_labels = all_reviews['Category'].tolist()

from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y_labels)
num_classes = len(label_encoder.classes_)
y = to_categorical(y_encoded, num_classes=num_classes)

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

vocab_size = 10000
max_length = 100
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(X_text)
X_sequences = tokenizer.texts_to_sequences(X_text)
X_padded = pad_sequences(X_sequences, maxlen=max_length, padding='post', truncating='post')
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_padded, y, test_size=0.2, random_state=42)
```

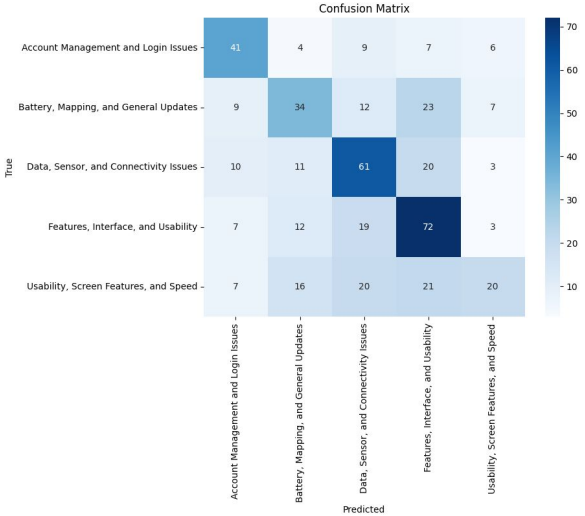
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=max_length))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

history = model.fit(
    X_train, y_train,
    epochs=15,
    batch_size=32,
    validation_split=0.2
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test Accuracy:", test_acc)
```

```
Epoch 1/15
/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_lengt
remove it.
  warnings.warn(
46/46 ━━━━━━━━━━━━━━━━━━━━ 3s 40ms/step - accuracy: 0.2575 - loss: 1.5954 - val_accuracy: 0.2637 - val_loss: 1.5690
Epoch 2/15
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.3009 - loss: 1.5484 - val_accuracy: 0.3764 - val_loss: 1.6360
Epoch 3/15
```

Results and Confusion Matrix

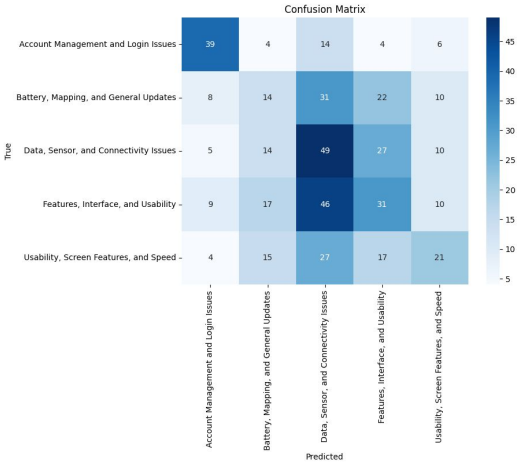
LSTM-based Model (Baseline)

Test Accuracy: 0.522



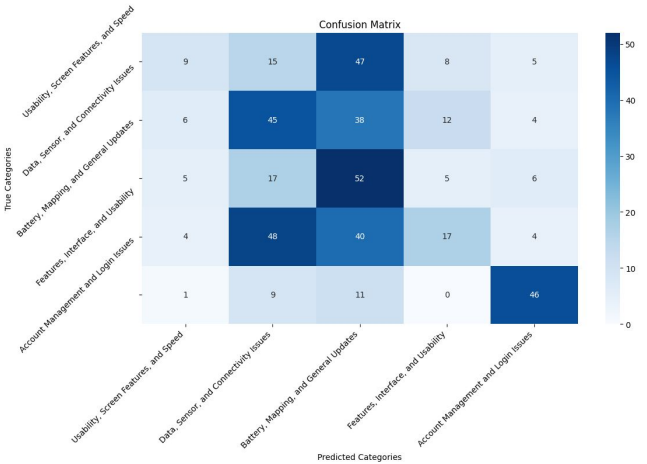
LSTM-based Model with Pretrained GloVe Embeddings

Test Accuracy: 0.3392



BERT-based Transformer Model

Test Accuracy: 0.3722



Pro/Cons

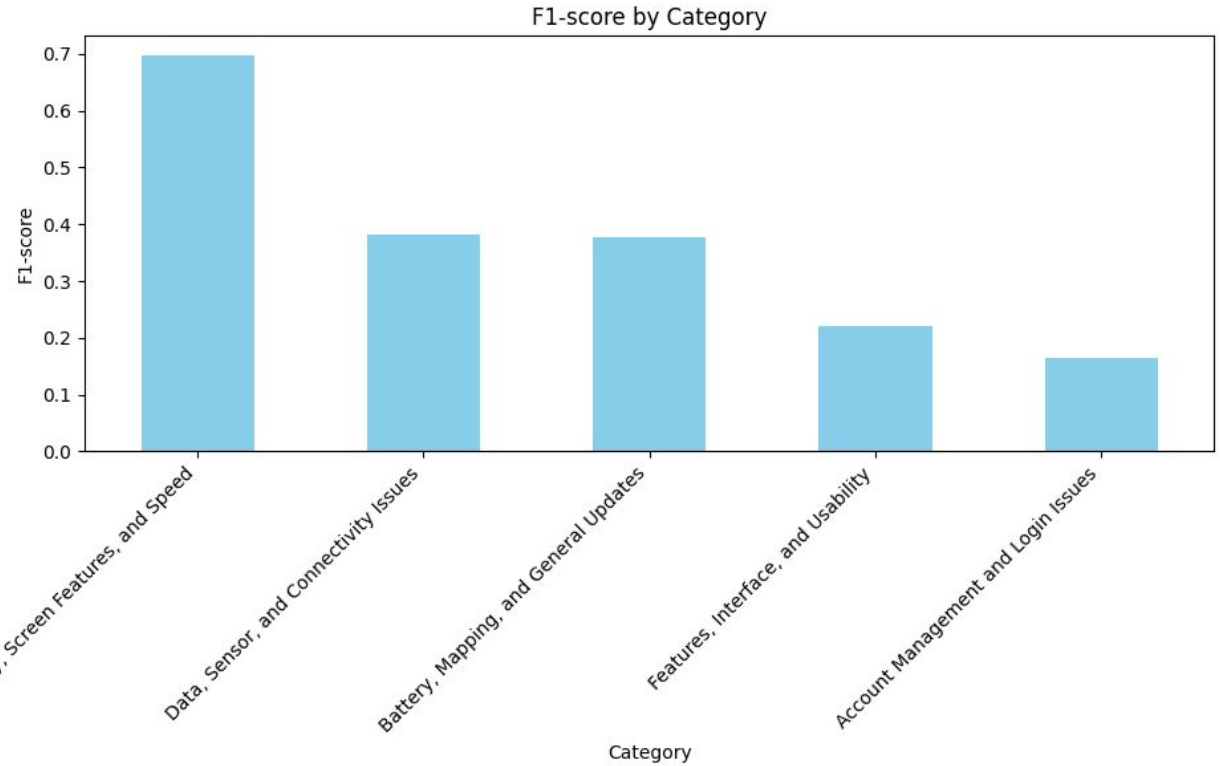
While I successfully implemented LDA for topic discovery and built a supervised classifier, the resulting accuracy was modest. Reasons include:

- Overlapping categories that are not easily separable by lexical features.
- Limited data or imbalance in categories.
- Complex user language and sarcasm not captured by simple embeddings.

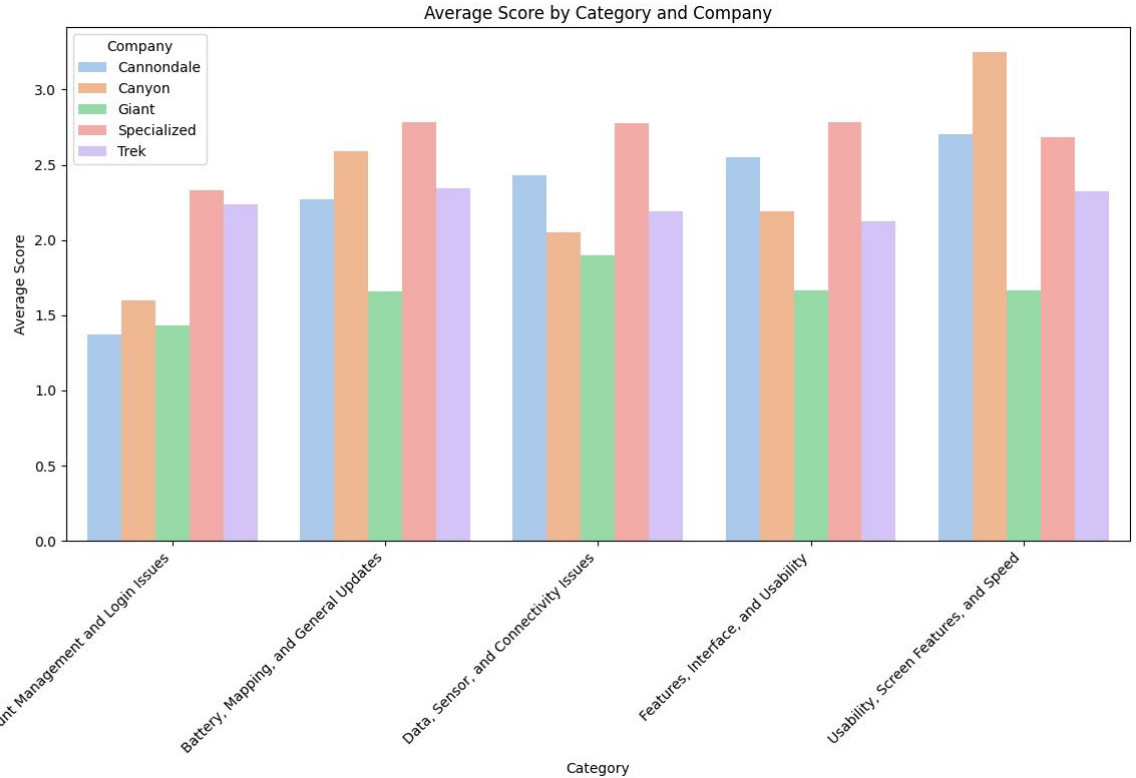
Future improvements might involve:

- Refining stopwords and preprocessing event further
- Collecting more data or trying other topic modeling techniques (like BERTopic).

Findings


F1-score by Category
F1-score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Usability, Screen Features, and Speed
Data, Sensor, and Connectivity Issues
Battery, Mapping, and General Updates
Features, Interface, and Usability
Account Management and Login Issues
Category

Findings



Category	Average Score
Account Management and Login Issues	1.656766
Battery, Mapping, and General Updates	2.343284
Data, Sensor, and Connectivity Issues	2.298969
Features, Interface, and Usability	2.273234
Usability, Screen Features, and Speed	2.325397

Conclusions

I demonstrated a pipeline from raw app reviews to meaningful categories and then trained a model to classify new reviews into these categories. Although performance is modest, this approach provides a framework for ongoing improvements. With enhanced methods and more data, such a system could guide e-bike companies in prioritizing feature improvements and addressing user concerns.

Youtube Link

https://youtu.be/Dap_Y_BqDpA