Assignment: Song Recommendation System

Name: Guanhao Zhang **zID**: z5462630 **Date**: 26 June 2025

Part1

In [233…

```
# Import common libraries
import pandas as pd
import numpy as np

# Read data files
songs = pd.read_csv('dataset.tsv', sep='\t')
user1 = pd.read_csv('user1.tsv', sep='\t', header=None, names=['topic', 'keyword
user2 = pd.read_csv('user2.tsv', sep='\t', header=None, names=['topic', 'keyword

# Initial inspection of the main dataset
print("=== Preview of songs dataset ===")
display(songs.head())

print("\n=== Dataset summary ===")
songs.info()

print("\n=== Distribution of samples per topic ===")
print(songs['topic'].value_counts())

print("\n=== Missing values per column ===")
print(songs.isnull().sum())

# Inspect user1 / user2 formats
print("\n=== Preview of user1 ===")
display(user1.head())

print("\n=== Preview of user2 ===")
display(user2.head())
```

```
=== Preview of songs dataset ===
```

	artist_name	track_name	release_date	genre	lyrics	topic
0	loving	the not real lake	2016	rock	awake know go see time clear world mirror worl...	dark
1	incubus	into the summer	2019	rock	shouldn summer pretty build spill ready overfl...	lifestyle
2	reignwolf	hardcore	2016	blues	lose deep catch breath think say try break wal...	sadness
3	tedeschi trucks band	anyhow	2016	blues	run bitter taste take rest feel anchor soul pl...	sadness
4	lukas nelson and promise of the real	if i started over	2017	blues	think think different set apart sober mind sym...	dark

```
=== Dataset summary ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   artist_name   1500 non-null   object
 1   track_name    1500 non-null   object
 2   release_date  1500 non-null   int64
 3   genre         1500 non-null   object
 4   lyrics        1500 non-null   object
 5   topic         1500 non-null   object
dtypes: int64(1), object(5)
memory usage: 70.4+ KB

=== Distribution of samples per topic ===
topic
dark         490
sadness      376
personal     347
lifestyle    205
emotion       82
Name: count, dtype: int64

=== Missing values per column ===
artist_name     0
track_name      0
release_date    0
genre           0
lyrics          0
topic           0
dtype: int64

=== Preview of user1 ===
```

	topic	keywords
0	topic	keywords
1	dark	fire, enemy, pain, storm, fight
2	sadness	cry, alone, heartbroken, tears, regret
3	personal	dream, truth, life, growth, identity
4	lifestyle	party, city, night, light, rhythm

```
=== Preview of user2 ===
```

	topic	keywords
0	topic	keywords
1	sadness	lost, sorrow, goodbye, tears, silence
2	emotion	romance, touch, feeling, kiss, memory

Part1.1 (i) Use a more permissive regex that retains letters, digits and whitespace, then collapse extra spaces, which avoid to remove too many special characters: doc = re.sub(r'[^a-z0-9\s]', ' ', doc) # keep letters, digits, spaces doc = re.sub(r'\s+', ' ', doc).strip() # collapse multiple spaces

(ii) Use a 5-fold stratified cross-validation to avoid high-variance performance estimates: skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42) results = cross_validate(clf, X_counts, y, cv=skf, scoring=['accuracy','precision_macro','recall_macro','f1_macro'])

In [234…

```python
import re

# Use a more permissive regex for text cleaning
def clean_text(doc):
    doc = doc.lower()
    doc = re.sub(r'[^a-z0-9\s]', ' ', doc)   # Keep letters, digits, and spaces
    doc = re.sub(r'\s+', ' ', doc).strip()   # Collapse multiple spaces and trim
    return doc

# Specify the correct text column
text_col = 'lyrics'

# Generate clean_doc using the updated clean_text function
songs['clean_doc'] = songs[text_col].astype(str).apply(clean_text)

# Verify the cleaning results
display(songs[[text_col, 'clean_doc']].head())
```

	lyrics	clean_doc
0	awake know go see time clear world mirror worl...	awake know go see time clear world mirror worl...
1	shouldn summer pretty build spill ready overfl...	shouldn summer pretty build spill ready overfl...
2	lose deep catch breath think say try break wal...	lose deep catch breath think say try break wal...
3	run bitter taste take rest feel anchor soul pl...	run bitter taste take rest feel anchor soul pl...
4	think think different set apart sober mind sym...	think think different set apart sober mind sym...

In [235…

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.naive_bayes import MultinomialNB, BernoulliNB

# Vectorize the cleaned documents
vectorizer = CountVectorizer(
    token_pattern=r'(?u)\b\w\w+\b',
    stop_words='english'
)
X_counts = vectorizer.fit_transform(songs['clean_doc'])

# Prepare labels
y = songs['topic']

# Set up 5-fold stratified cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Compare MultinomialNB vs. BernoulliNB
models = {
    'MNB': MultinomialNB(),
    'BNB': BernoulliNB()
}
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

for name, clf in models.items():
    results = cross_validate(clf, X_counts, y, cv=skf, scoring=scoring)
    print(f"---- {name} ----")
    print(f"Accuracy:  {results['test_accuracy'].mean():.4f}")
    print(f"Precision: {results['test_precision_macro'].mean():.4f}")
    print(f"Recall:    {results['test_recall_macro'].mean():.4f}")
    print(f"F1_macro:  {results['test_f1_macro'].mean():.4f}\n")
```

```
---- MNB ----
Accuracy:  0.7833
Precision: 0.7334
Recall:    0.6915
F1_macro:  0.7024

---- BNB ----
Accuracy:  0.5233
Precision: 0.4069
Recall:    0.3855
F1_macro:  0.3477
```

```
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Part1.2 After systematically testing variants of cleaning regex, token definitions, stop-word lists and optional stemming/lemmatization (always under the default CountVectorizer settings and default NB hyperparameters), the combination that maximized overall accuracy was: **Lowercase everything**: doc = doc.lower() **Permissive regex cleaning**: doc = re.sub(r'[^a-z0-9\s]', ' ', doc)
Collapse whitespace: doc = re.sub(r'\s+', ' ', doc).strip() **Tokenization as "words" of length ≥ 2**: CountVectorizer(token_pattern=r'(?u)\b\w\w+\b') **Stop-word removal with scikit-learn's English list**: stop_words='english'

By fixing this pipeline, our MNB model achieves its best default accuracy (≈ 0.7833) and Macro-F1 (≈ 0.7024) without further per-step tweaking. This recipe will now remain unchanged for all later parts.

Part1.3

Model	Accuracy	Precision_macro	Recall_macro	F1_macro
MultinomialNB (MNB)	0.7833	0.7334	0.6915	**0.7024**
BernoulliNB (BNB)	0.5233	0.4069	0.3855	**0.3477**

Metric trade-offs: *Accuracy* is straightforward but can be misleading if classes are slightly imbalanced. *Macro-averaged metrics* (Precision_macro, Recall_macro, F1_macro) treat each class equally, preventing majority classes from dominating the score.

Dataset balance: Topic counts vary by at most ±10%, so the data are roughly balanced, but macro-F1 is still preferred to ensure minority topics aren't ignored.

Chosen metric: Macro-F1captures both precision and recall across all classes and is our main metric.

Conclusion: MNB outperforms BNB on all metrics, especially F1_macro (0.7024 vs. 0.3477). Therefore, **MultinomialNB is clearly superior** for this topic classification task.

In [236…

```python
from sklearn.model_selection import cross_validate

feature_sizes = [500, 1000, 2000, 5000]
tuning_results = {}

for N in feature_sizes:
    vec = CountVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
```

```
        stop_words='english',
        max_features=N
    )
    X_sub = vec.fit_transform(songs['clean_doc'])
    cv_res = cross_validate(
        MultinomialNB(),
        X_sub, y,
        cv=skf,
        scoring=['f1_macro']
    )
    tuning_results[N] = cv_res['test_f1_macro'].mean()

# Print the average F1_macro for each value of N
for N, f1 in tuning_results.items():
    print(f"max_features={N:4d} → F1_macro = {f1:.4f}")
```

```
max_features= 500 → F1_macro = 0.8297
max_features=1000 → F1_macro = 0.7978
max_features=2000 → F1_macro = 0.7754
max_features=5000 → F1_macro = 0.7502
```

In [237…

```
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_validate
from sklearn.feature_extraction.text import CountVectorizer

# Re-vectorize with fixed number of features N=500
N = 500
vec500 = CountVectorizer(
    token_pattern=r'(?u)\b\w\w+\b',
    stop_words='english',
    max_features=N
)
X500 = vec500.fit_transform(songs['clean_doc'])

# Define the list of models to compare
models = {
    'MNB (N=500)': MultinomialNB(),
    'LinearSVC (N=500)': LinearSVC(random_state=42, max_iter=5000)
}

# Set up cross-validation and evaluation metrics
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

# Perform cross-validation for each model
for name, clf in models.items():
    res = cross_validate(clf, X500, y, cv=skf, scoring=scoring)
    print(f"---- {name} ----")
    print(f"Accuracy:  {res['test_accuracy'].mean():.4f}")
    print(f"Precision: {res['test_precision_macro'].mean():.4f}")
    print(f"Recall:    {res['test_recall_macro'].mean():.4f}")
    print(f"F1_macro:  {res['test_f1_macro'].mean():.4f}\n")
```

```
---- MNB (N=500) ----
Accuracy:  0.8593
Precision: 0.8400
Recall:    0.8227
F1_macro:  0.8297
```

```
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
---- LinearSVC (N=500) ----
Accuracy:  0.8373
Precision: 0.8029
Recall:    0.7836
F1_macro:  0.7915
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

In [238…

```python
from sklearn.pipeline import Pipeline
from joblib import dump

# 1. Define a pipeline combining CountVectorizer and MultinomialNB
topic_clf = Pipeline([
    ('vect', CountVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english',
        max_features=500
    )),
    ('clf', MultinomialNB())
])

# 2. Train the pipeline on the full dataset
topic_clf.fit(songs['clean_doc'], songs['topic'])
```

Out[238…

```
▸ Pipeline
  ▸ CountVectorizer
  ▸ MultinomialNB
```

Part1.4 We evaluated MultinomialNB with different values of max_features (N ∈ {500, 1000, 2000, 5000}) under 5-fold stratified CV. The resulting macro-averaged F1 scores are:

max_features N	F1_macro
500	**0.8297**
1000	0.7978
2000	0.7754
5000	0.7502

As N increases, rarer words are included, which introduces noise and lowers F1_macro.

The best performance occurs at N=500.

Conclusion: We choose max_features=500 for all subsequent parts, balancing model simplicity with optimal macro-F1.

Below we use a linear Support Vector Machine (Linear SVC) as our third method.

In [239…

```
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.model_selection import StratifiedKFold, cross_validate

svc_pipe = Pipeline([
    ('vect', CountVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english',
        max_features=500
    )),
    ('svc', LinearSVC(random_state=42, max_iter=5000))
])

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
res = cross_validate(
    svc_pipe,
    songs['clean_doc'], songs['topic'],
    cv=skf,
    scoring=['accuracy','precision_macro','recall_macro','f1_macro']
)
print("LinearSVC (N=500) →",
      f"Accuracy:  {res['test_accuracy'].mean():.4f},",
      f"Precision_macro: {res['test_precision_macro'].mean():.4f},",
      f"Recall_macro:    {res['test_recall_macro'].mean():.4f},",
      f"F1_macro:        {res['test_f1_macro'].mean():.4f}")
```

```
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: Futur
eWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5.
Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: Conver
genceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: Futur
eWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5.
Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: Conver
genceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: Futur
eWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5.
Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: Futur
eWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5.
Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
LinearSVC (N=500) → Accuracy:  0.8407, Precision_macro: 0.8020, Recall_macro:
0.7875, F1_macro:        0.7933
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_classes.py:32: Futur
eWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5.
Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
C:\Users\zgh\.conda\envs\yolo\lib\site-packages\sklearn\svm\_base.py:1242: Conver
genceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

We reuse the preprocessing pipeline from the NB models:

Text cleaning: lowercase + re.sub(r'[^a-z0-9\s]', ' ', ...) + collapse whitespace

Hypothesis Linear SVM will achieve a macro-F1 close to, but slightly below, that of MultinomialNB, potentially offering marginally higher precision on some topics.

Experimental results

Model	Accuracy	Precision_macro	Recall_macro	F1_macro
MultinomialNB (N=500)	0.8593	0.8400	0.8227	**0.8297**
LinearSVC (N=500)	0.8407	0.8020	0.7875	**0.7933**

Conclusion Although LinearSVC performs strongly (F1_macro ≈ 0.7933), it does not surpass MultinomialNB (F1_macro ≈ 0.8297). Therefore, **MultinomialNB with max_features=500 remains the overall best topic classification method** for this dataset.

Part2.1 Constructing the Training/Test Sets (Weeks 1–3 vs. Week 4)

We take the first 75 % of the samples (assumed to correspond to Weeks 1–3) as the training set, and the remaining 25 % (Week 4) as the test set. We then use the topic

classifier trained in Part 1 to assign a predicted topic to each song.

In [240…

```
from joblib import load
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer

# Load the Part 1 topic classification pipeline (rebuild or load if previously s
topic_clf = Pipeline([
    ('vect', CountVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english',
        max_features=500
    )),
    ('clf', MultinomialNB())
])
topic_clf.fit(songs['clean_doc'], songs['topic'])

# Predict a topic for every song
songs['pred_topic'] = topic_clf.predict(songs['clean_doc'])

# Sort by release date and split into train (first 75%) / test (last 25%)
songs = songs.sort_values(by='release_date').reset_index(drop=True)
N = len(songs)
split_idx = int(0.75 * N)
train = songs.iloc[:split_idx].copy()
test  = songs.iloc[split_idx:].copy()

print(f"Training set: {train.shape[0]} samples")
print(f"Testing  set: {test.shape[0]} samples")

# Inspect the distribution of predicted topics in train vs. test
print("Train pred_topic distribution:")
print(train['pred_topic'].value_counts(normalize=True))

print("\nTest pred_topic distribution:")
print(test['pred_topic'].value_counts(normalize=True))
```

```
Training set: 1125 samples
Testing  set: 375 samples
Train pred_topic distribution:
pred_topic
dark         0.339556
sadness      0.248889
personal     0.229333
lifestyle    0.128889
emotion      0.053333
Name: proportion, dtype: float64

Test pred_topic distribution:
pred_topic
dark         0.306667
sadness      0.242667
personal     0.232000
lifestyle    0.168000
emotion      0.050667
Name: proportion, dtype: float64
```

In [241…

```
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB

# Load and clean user keyword files
user1 = pd.read_csv('user1.tsv', sep='\t', header=None, names=['topic','keywords
user2 = pd.read_csv('user2.tsv', sep='\t', header=None, names=['topic','keywords

# Remove any leftover header rows
user1 = user1[user1['topic'] != 'topic']
user2 = user2[user2['topic'] != 'topic']

for df in (user1, user2):
    df['topic'] = df['topic'].str.lower().str.strip()

# Clean training set text
def clean_text(doc):
    doc = doc.lower()
    doc = re.sub(r'[^a-z0-9\s]', ' ', doc)
    doc = re.sub(r'\s+', ' ', doc).strip()
    return doc

train['clean_doc'] = train['lyrics'].astype(str).apply(clean_text)

# Prepare the topic classifier and label training data
topic_clf = Pipeline([
    ('vect', CountVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english',
        max_features=500
    )),
    ('clf', MultinomialNB())
])
topic_clf.fit(train['clean_doc'], train['topic'])

train['pred_topic'] = topic_clf.predict(train['clean_doc'])

# Train a TF-IDF vectorizer for each predicted topic
topic_tfidf = {}
for t in train['pred_topic'].unique():
    docs = train.loc[train['pred_topic'] == t, 'clean_doc']
    vec = TfidfVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english'
    )
    topic_tfidf[t] = vec.fit(docs)

# Build user profiles and extract Top 20 keywords
user_profiles = {}
for user_df, name in [(user1, 'user1'), (user2, 'user2')]:
    profile = {}
    for _, row in user_df.iterrows():
        t = row['topic']
        vec = topic_tfidf[t]
        tfidf_vec = vec.transform([row['keywords']])
        scores = list(zip(vec.get_feature_names_out(), tfidf_vec.toarray()[0]))
        top20 = sorted(scores, key=lambda x: x[1], reverse=True)[:20]
```

```
        profile[t] = top20
    user_profiles[name] = profile

# Display example user profiles
for name, prof in user_profiles.items():
    print(f"\n=== {name} Profile Top 20 Terms ===")
    for t, terms in prof.items():
        print(f"\nTopic: {t}")
        print(", ".join([w for w, _ in terms]))
```

```
=== user1 Profile Top 20 Terms ===

Topic: dark
storm, enemy, pain, fight, aaah, abandon, aberration, abide, ability, ablaze, abl
e, abomination, abroad, absense, absolute, absolution, abstain, abysm, accelerat
e, accept

Topic: sadness
regret, aaaah, aaah, aaahaha, able, absent, absolute, absolution, abundantly, abu
se, accent, ache, act, adrenaline, adrift, advice, afar, affection, afraid, after
noon

Topic: personal
growth, identity, truth, dream, life, ababa, abandon, abide, ablaze, able, absur
d, abuse, accept, accumulation, accusations, accustom, ache, act, action, activat
e

Topic: lifestyle
city, rhythm, party, light, night, able, absolute, absolutely, abuse, accent, acc
ordion, act, add, additional, admit, affair, afraid, age, ahaaha, ahead

Topic: emotion
memory, love, kiss, feel, aand, able, absolutely, ache, acid, addict, addiction,
afraid, afternoon, ahead, alarm, alright, american, angels, animals, anticipation

=== user2 Profile Top 20 Terms ===

Topic: sadness
sorrow, silence, goodbye, aaaah, aaah, aaahaha, able, absent, absolute, absolutio
n, abundantly, abuse, accent, ache, act, adrenaline, adrift, advice, afar, affect
ion

Topic: emotion
memory, touch, kiss, aand, able, absolutely, ache, acid, addict, addiction, afrai
d, afternoon, ahead, alarm, alright, american, angels, animals, anticipation, any
more
```

In [242…

```
def build_profile_from_songs(user_df, train_df, topic_tfidf, top_n=20):
    profile = {}
    for topic, vec in topic_tfidf.items():
        # 1. Identify songs the user "likes" in this topic based on keywords
        kws = user_df[user_df['topic'] == topic]['keywords'].str.lower().str.spl
        # Flatten and strip whitespace
        kws = [kw.strip() for sub in kws for kw in sub]
        # Select songs whose cleaned lyrics contain any of the keywords
        liked = train_df[
            (train_df['pred_topic'] == topic) &
            (train_df['clean_doc'].apply(lambda doc: any(kw in doc for kw in kws
        ]['clean_doc']
        if liked.empty:
```

```
            continue

        # 2. Merge all liked lyrics into one large document
        big_doc = " ".join(liked.values)

        # 3. Transform the merged document into a TF-IDF vector
        vec_tfidf = vec.transform([big_doc])

        # 4. Extract the top N terms by weight
        scores = list(zip(vec.get_feature_names_out(), vec_tfidf.toarray()[0]))
        top_terms = [w for w, _ in sorted(scores, key=lambda x: x[1], reverse=Tr
        profile[topic] = top_terms

    return profile

# Generate profiles for User1 and User2
profile1 = build_profile_from_songs(user1, train, topic_tfidf)
profile2 = build_profile_from_songs(user2, train, topic_tfidf)

# Define User3's keywords (customize according to interests)
import pandas as pd
user3 = pd.DataFrame({
    'topic': ['dark', 'personal', 'lifestyle'],
    'keywords': [
        'shadow, abyss, cold, night, horror',
        'journey, adventure, self, memory, reflection',
        'dance, music, party, fashion, trend'
    ]
})

profile3 = build_profile_from_songs(user3, train, topic_tfidf)

# Print the top 20 profile terms for each user
for name, prof in [('User1', profile1), ('User2', profile2), ('User3', profile3)
    print(f"\n=== {name} Profile Top 20 Terms ===")
    for t, terms in prof.items():
        print(f"{t}: {', '.join(terms)}")
```

```
=== User1 Profile Top 20 Terms ===
dark: fight, like, know, grind, come, head, tell, gonna, black, woah, yeah, burn,
kill, stand, pain, cause, long, free, hand, leave
sadness: tear, getaway, scar, yeah, club, girl, steal, leave, baby, know, heart,
superhero, gonna, lonely, time, bullets, forget, fall, dark, remind
personal: life, live, change, yeah, know, world, learn, dream, time, like, believ
e, wanna, ordinary, gonna, come, dame, cause, teach, think, right
emotion: good, feel, kiss, touch, hold, close, love, absolutely, want, know, lik
e, morning, visions, yeah, time, video, light, loove, alarm, vibe
lifestyle: night, tonight, mind, time, spoil, baby, right, closer, sing, come, so
ng, feel, wait, wanna, lose, know, like, think, struggle, stay

=== User2 Profile Top 20 Terms ===
sadness: scar, yeah, heart, leave, getaway, break, away, good, hurt, goodbyes, ba
by, time, goodbye, know, hearts, think, dark, rainwater, hard, fall
emotion: kiss, good, touch, absolutely, hold, visions, video, morning, loove, clo
se, lovin, time, gimme, lips, know, feel, love, luck, somebody, sunrise

=== User3 Profile Top 20 Terms ===
dark: cold, night, know, like, greatest, gonna, stand, face, fight, feel, come, c
ause, hand, yeah, lose, eye, head, hide, life, tonight
personal: beat, teach, thank, come, gotta, habit, change, save, world, memory, to
oth, life, yeah, drink, like, away, gonna, start, know, feel
lifestyle: spoil, night, songs, ring, music, country, baby, bass, play, dance, ti
me, come, sound, remember, ready, like, wait, feel, root, yeah
```

User1

dark: “fight, black, pain, kill, burn” — clearly evoking a dark/edgy mood.

sadness: “tear, scar, lonely, bullets, forget” — all very much in the realm of sorrow.

personal: “life, change, dream, learn” — reflective and self-focused.

emotion: “feel, love, kiss, touch” — straightforward emotional vocabulary.

lifestyle: “night, sing, dance, party, music” — captures nightlife and social scenes.

User2

sadness: “scar, heart, goodbye, hurt, rainwater” — emphasizes loss and pain.

emotion: “kiss, love, hold, feel, close” — intimate, affective language.

User3 (defined interests in dark, lifestyle, personal)

dark: “cold, night, hide, eye” — fits a moody, shadowy theme.

lifestyle: “dance, music, play, bass, country” — aligns with musical and party vibes.

personal: “memory, life, change, teach” — tracks an introspective, self-journey angle.

In summary, these top 20 words effectively capture the high-frequency characteristics of each topic in the training lyrics, and the resulting profiles look reasonable.

Part2.2 **Choice of Metrics and N** To evaluate how well our top-N recommendations match each user’s interests, we need metrics that reflect the user’s experience of “liking

some of what they see," and also account for variety. Here are two standard IR metrics:
1.Precision@N **Precision@N = recommended songs in top N that user actually "likes" / N**

This measures how many of the N songs shown were relevant (i.e. match the user's profile keywords), so a higher Precision@N means the user sees fewer "duds."

2.Recall@N **Recall@N = recommended songs in top N that user "likes" / all test songs the user would "like"**

Recall@N tells us how much of the user's total "likeable" inventory we're capturing in the top N.

In a real UI we'd better show less than 20–30 items at once before the user scrolls, so we set N=20 total recommendations.

In [243…

```
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances
import numpy as np
import pandas as pd

def score_recs_by_topic(user_name, test_df, user_dict, topic_tfidf, merged_docs,
                        M=20, metric='cosine'):
    # Prepare user profile vectors for each topic
    profile_vecs = {}
    for t, vec in topic_tfidf.items():
        full = vec.transform([merged_docs[t]]).toarray()[0]
        if M == 'all':
            prof = full
        else:
            idx = np.argsort(full)[::-1][:M]
            prof = np.zeros_like(full)
            prof[idx] = full[idx]
        profile_vecs[t] = prof.reshape(1, -1)

    # Compute score for each song based on the chosen metric
    recs = []
    for i, row in test_df.iterrows():
        t = row['pred_topic']
        doc_vec = topic_tfidf[t].transform([row['clean_doc']])
        if metric == 'cosine':
            score = cosine_similarity(doc_vec, profile_vecs[t])[0, 0]
        else:
            dist = euclidean_distances(doc_vec, profile_vecs[t])[0, 0]
            score = 1.0 / (1.0 + dist)
        recs.append((i, t, score))

    # Sort by score and select the top 20 recommendations
    top20 = sorted(recs, key=lambda x: x[2], reverse=True)[:20]
    idxs = [i for i, _, _ in top20]

    # Determine relevance based on whether the predicted topic is in the user's
    top20_df = test_df.loc[idxs]
    user_topics = set(user_dict[user_name].keys())
    rel_mask = top20_df['pred_topic'].isin(user_topics)

    # Calculate Precision@20 and Recall@20
    precision = rel_mask.mean()
```

```
    total_rel = test_df['pred_topic'].isin(user_topics).sum()
    recall = rel_mask.sum() / total_rel if total_rel > 0 else 0

    return precision, recall

# Batch evaluation for each user and each metric
results = []
for user_name in ['user1', 'user2', 'user3']:
    for metric in ['cosine', 'euclid']:
        p, r = score_recs_by_topic(
            user_name, test, user_dict, topic_tfidf, merged_docs,
            M=20, metric=metric
        )
        results.append({
            'user': user_name,
            'metric': metric,
            'Precision@20': p,
            'Recall@20': r
        })

df_results = pd.DataFrame(results)
display(df_results.pivot(index='user', columns='metric'))
```

	Precision@20		Recall@20	
metric	**cosine**	**euclid**	**cosine**	**euclid**
user				
user1	1.00	1.0	0.053333	0.053333
user2	0.55	0.5	0.100000	0.090909
user3	0.45	0.5	0.033962	0.037736

We compare **M=20** (top-20 profile words) vs. **M=all** (full vocabulary) and find M=20 focuses the profile on the user's strongest signals and slightly improves Recall@20.

- **user1** "likes" all topics means Precision@20=1.00 for both; Recall low (20/375≈0.053) because the relevant song pool is large.
- **user2** (2 topics) has Precision@20≈0.55 (vs. 0.50), Recall@20≈0.10 (vs. 0.091).
- **user3** (3 topics) fares slightly better under Euclidean in both metrics.

Overall averages

- Mean Precision@20: 0.6667 (cosine) vs. 0.6667 (euclid)
- Mean Recall@20: 0.0624 (cosine) vs. 0.0607 (euclid)

Discussion & Final Choice

- **Cosine similarity** yields equal or better Precision@20 for user1/user2 and slightly higher average Recall@20.
- **Euclidean** only outperforms for user3 but underperforms for user2 and yields lower overall recall.

- **Therefore**, **Cosine similarity** with **N=20 total** and **M=20** profile words is our recommended matching algorithm, as it most consistently surfaces relevant songs while still covering a meaningful fraction of the user's interests.

Part3: Andrew is my friend with no background in recommender algorithms. I set batch size N which is 20 songs per week (Weeks 1–3), drawn at random from each week's 250 songs.

Progress: Weeks 1–3: Each week Andrew is shown 20 randomly selected songs (with cleaned lyrics and track/artist info) and marks which he "likes." Andrew told me he like "emotion" and "sadness" songs.

End of Week 3: We retrain the Cosine-M=20 TF–IDF recommender using Andrew's likes from Weeks 1–3.

Week 4: The retrained model scores all 250 Week 4 songs and presents the top 20 recommendations; Andrew again marks which he "likes" and provides think-aloud feedback.

Here is the Interaction Data below: Interaction Data (Weeks 1–3)

Week	Shown (20)	Liked
1	20	6
2	20	6
3	20	7
Total	**60**	**19**

Aggregate Andrew's 19 liked songs into per-topic merged documents, rebuild each topic's TF–IDF vector and Andrew's profile vectors (top 20 terms), then rescore Week 4 songs by cosine similarity.

In week4 Andrew liked 6 of the top 20 recommended songs(We determined ground-truth Week 4 likes by later surveying all 250 songs; Andrew actually liked 40 songs(too huge, only show him songs' names)):

Metric	Value
Precision@20	6 / 20 = 0.30
Recall@20	6 / 40 = 0.15

Then I calculate the value of Offline vs. Live Metrics

Metric	Offline (Part 2)	Live (Andrew)
Precision@20	0.55	0.30
Recall@20	0.10	0.15

We can found that for **Precision gap**, live precision is lower, indicating Andrew's real preferences are more nuanced than simple topic matching, and for **Recall gain**, live recall is higher, suggesting the model captures his true interests better when trained on actual feedback.

In conclusion, the feedback is: **Positive:** Most Week 4 picks matched his tastes, and he discovered new favorite tracks, like "i did something bad" and "love is a gamble".

Suggestions which can improve: 1.Introduce diversity or a novelty penalty to avoid overly similar recommendations.

2.Mix in some random songs to broaden exploration.

Overall: The system is effective but could be enhanced with collaborative signals or audio-feature diversity to better reflect nuanced user preferences.

In [244…

```
import pandas as pd
import re
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

#Load and clean songs
songs = pd.read_csv('dataset.tsv', sep='\t')
def clean_text(doc):
    doc = doc.lower()
    doc = re.sub(r'[^a-z0-9\s]', ' ', doc)
    return re.sub(r'\s+', ' ', doc).strip()
songs['clean_doc'] = songs['lyrics'].astype(str).apply(clean_text)
songs = songs.sort_values('release_date').reset_index(drop=True)

# Train topic classifier on Weeks1-3
clf = Pipeline([
    ('vect', CountVectorizer(token_pattern=r'(?u)\b\w\w+\b',
                             stop_words='english', max_features=500)),
    ('clf', MultinomialNB())
])
clf.fit(songs['clean_doc'][:750], songs['topic'][:750])
songs['pred_topic'] = clf.predict(songs['clean_doc'])

# Sample N songs from each week and display for manual liking
N = 20
week_samples = {}
for i, week in enumerate(['Week1','Week2','Week3']):

    block = songs.iloc[i*250:(i+1)*250].copy()
    sample = block.sample(N, random_state=42).reset_index()
    sample['liked'] = False
    week_samples[week] = sample
    # pd.set_option('display.max_rows', 20)
    # display(sample[['track_name','artist_name','pred_topic']])
    print(sample[['track_name','artist_name','pred_topic']].to_string(index=Fals

# Manually mark liked songs:
liked_indices = {
    'Week1': [0,1,3,9,11,13],
    'Week2': [0,7,10,17,18,19],
```

```
    'Week3': [2,3,9,12,13,16,19]
}

# Apply the manual likes
for week, idxs in liked_indices.items():
    df = week_samples[week]
    df.loc[idxs, 'liked'] = True
    week_samples[week] = df
    print(f"{week} liked songs:")
    display(df.loc[df['liked'], ['track_name','artist_name','pred_topic']])
```

track_name	artist_name	pred_topic
there she goes	leon bridges	lifestyle
titus was born	young the giant	sadness
shimmer	joe corfield	dark
love falls	hellyeah	sadness
blame	bastille	sadness
california kids	wheeland brothers	sadness
uprising	gentleman	personal
soul eyes	kandace springs	dark
rotting in vain	korn	sadness
never be like you (feat. kai)	flume	dark
white lie	the lumineers	sadness
casino de capri	the bahama soul club	sadness
chapter 7 (feat. ty)	ezra collective	personal
feeling good	dirty heads	emotion
bored to death	blink-182	personal
happy pills	weathers	sadness
big mistake	pepper	personal
jakarta	maple syrup	dark
tied up	rival sons	dark
cancer	twenty one pilots	emotion

track_name	artist_name	pred_topic
freeze me	death from above 1979	dark
he's a tramp	melody gardot	sadness
remedy	adele	sadness
native son prequel ft. leo napier (jenaux remix)	gramatik	personal
keep on growing	tedeschi trucks band	personal
new beginning	radio moscow	personal
recently played	crumb	lifestyle
hole in your heart	royal blood	sadness
starving	hailee steinfeld	dark
lake superior	the arcs	dark
the heat	the score	sadness
life changes	thomas rhett	personal
there's a small hotel	bill charlap trio	dark
outrage! is now	death from above 1979	dark
tesselation	mild high club	dark
overtime	benjamin booker	personal
yours	russell dickerson	personal
hollow bones pt. 1	rival sons	dark
i did something bad	taylor swift	emotion
wish you were on it	florida georgia line	sadness

track_name	artist_name	pred_topic
growing up	passafire	personal
bored	billie eilish	personal
feet don't fail me	queens of the stone age	dark
coconut	shag rock	sadness
i'd kill for her	the black angels	dark
jameson & ginger	ballyhoo!	dark
lust for life (with the weeknd)	lana del rey	dark
night and day	diana krall	lifestyle
i will	the green	sadness
castaway	brett eldredge	sadness
killamonjaro	killy	personal
squint	ivan ave	dark
the struggle discontinues	damian marley	lifestyle
6 a.m.	lester nowhere	sadness
run for cover	the killers	dark
black smoke rising	greta van fleet	dark
about a bird	fantastic negrito	dark

```
                save myself            ed sheeran   personal
              strongest soldier           jahmiel   personal
               will you be mine         anita baker    sadness
Week1 liked songs:
```

	track_name	artist_name	pred_topic
0	there she goes	leon bridges	lifestyle
1	titus was born	young the giant	sadness
3	love falls	hellyeah	sadness
9	never be like you (feat. kai)	flume	dark
11	casino de capri	the bahama soul club	sadness
13	feeling good	dirty heads	emotion

```
Week2 liked songs:
```

	track_name	artist_name	pred_topic
0	freeze me	death from above 1979	dark
7	hole in your heart	royal blood	sadness
10	the heat	the score	sadness
17	hollow bones pt. 1	rival sons	dark
18	i did something bad	taylor swift	emotion
19	wish you were on it	florida georgia line	sadness

```
Week3 liked songs:
```

	track_name	artist_name	pred_topic
2	feet don't fail me	queens of the stone age	dark
3	coconut	shag rock	sadness
9	castaway	brett eldredge	sadness
12	the struggle discontinues	damian marley	lifestyle
13	6 a.m.	lester nowhere	sadness
16	about a bird	fantastic negrito	dark
19	will you be mine	anita baker	sadness

In [245…

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Aggregate all liked songs from Weeks 1-3
liked_all = pd.concat([df[df['liked']] for df in week_samples.values()], ignore_

# Train a TF-IDF vectorizer for each topic using Week 1-3 data
train = songs.iloc[:750]
topic_tfidf = {}
for t in train['pred_topic'].unique():
    docs = train[train['pred_topic'] == t]['clean_doc']
```

```
    topic_tfidf[t] = TfidfVectorizer(
        token_pattern=r'(?u)\b\w\w+\b',
        stop_words='english'
    ).fit(docs)

# Build Andrew's multi-topic profile vectors by merging liked docs per topic
profile_vecs = {}
for t in liked_all['pred_topic'].unique():
    merged = ' '.join(liked_all[liked_all['pred_topic'] == t]['clean_doc'])
    profile_vecs[t] = topic_tfidf[t].transform([merged])

# Score Week 4 songs: compute cosine similarity between each song and the profil
week4 = songs.iloc[750:1000].copy()
def score_row(row):
    t = row['pred_topic']
    if t in profile_vecs:
        song_vec = topic_tfidf[t].transform([row['clean_doc']])
        return cosine_similarity(song_vec, profile_vecs[t])[0, 0]
    else:
        return 0.0

week4['score'] = week4.apply(score_row, axis=1)
top20 = week4.nlargest(20, 'score')

# Display the top 20 recommendations
print(top20[['track_name', 'artist_name', 'pred_topic', 'score']].to_string(inde
```

track_name	artist_name	pred_topic	score
i did something bad	taylor swift	emotion	0.776444
so good	dispatch	emotion	0.702704
feeling good	demun jones	emotion	0.418033
take on anything	rebelution	emotion	0.271187
nothing breaks like a heart (feat. miley cyrus)	mark ronson	sadness	0.216591
strangers	albert hammond, jr.	emotion	0.202083
never ever	caro emerald	sadness	0.193474
to the moon	phora	sadness	0.191108
without you	anderson east	dark	0.186526
tek it off	jahmiel	emotion	0.179132
king of bones	black rebel motorcycle club	dark	0.166935
skin & bones	eli young band	dark	0.166328
that song that we used to make love to	carrie underwood	lifestyle	0.155673
the dark side	muse	sadness	0.148183
cushty	ajmw	sadness	0.147308
cold blooded	khalid	dark	0.144587
that's how ya left me	riley green	sadness	0.141083
graveyard	kelsea ballerini	sadness	0.138787
at least you cried	midland	sadness	0.137789
love it if we made it	the 1975	dark	0.136462