APR Assignment: Anime Score Classification

Name: Sama Supratheek Reddy
Roll No: 2201CS62

Objective

Classify anime shows into score categories (`Very Good` , `Good` , `Average` , `Low`) based on features such as episodes, duration, genres, producers, studios, licensors, source material, and rating.

In [15]:
```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer, OneHotEncoder, StandardScale
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import sklearn

print("scikit-learn version:", sklearn.__version__)
```

```
scikit-learn version: 1.7.2
```

In [16]:
```
df = pd.read_csv("anime.csv")
print("Initial dataset shape:", df.shape)
print("Columns:", df.columns.tolist())
```

```
Initial dataset shape: (17562, 35)
Columns: ['MAL_ID', 'Name', 'Score', 'Genres', 'English name', 'Japanese name', 'Typ
e', 'Episodes', 'Aired', 'Premiered', 'Producers', 'Licensors', 'Studios', 'Source',
'Duration', 'Rating', 'Ranked', 'Popularity', 'Members', 'Favorites', 'Watching', 'C
ompleted', 'On-Hold', 'Dropped', 'Plan to Watch', 'Score-10', 'Score-9', 'Score-8',
'Score-7', 'Score-6', 'Score-5', 'Score-4', 'Score-3', 'Score-2', 'Score-1']
```

In [17]:
```
df['Score'] = pd.to_numeric(df['Score'], errors='coerce')
df = df.dropna(subset=['Score']).reset_index(drop=True)

def bucket_score(s):
    if s >= 8.0: return "Very Good"
    elif s >= 7.0: return "Good"
    elif s >= 6.0: return "Average"
    else: return "Low"

df['Score_Class'] = df['Score'].apply(bucket_score)
```

```
score_cols = [c for c in df.columns if c.lower().startswith('score-') or c == 'Scor
df_features = df.drop(columns=score_cols)

print("Dropped columns:", score_cols)
print("Target class distribution:\n", df['Score_Class'].value_counts())
```

```
Dropped columns: ['Score', 'Score-10', 'Score-9', 'Score-8', 'Score-7', 'Score-6',
'Score-5', 'Score-4', 'Score-3', 'Score-2', 'Score-1']
Target class distribution:
 Score_Class
Average      5300
Low          3341
Good         3232
Very Good     548
Name: count, dtype: int64
```

In [18]:

```
df_use = df_features.copy()

df_use['Episodes'] = pd.to_numeric(df_use['Episodes'], errors='coerce')
df_use['Episodes'].fillna(df_use['Episodes'].median(), inplace=True)

def parse_duration_to_min(x):
    if pd.isna(x): return np.nan
    s = str(x)
    try:
        hours = mins = 0
        if "hr" in s:
            parts = s.split()
            for i, token in enumerate(parts):
                if token.isdigit() and i+1 < len(parts) and parts[i+1].startswith("
                if token.isdigit() and i+1 < len(parts) and parts[i+1].startswith("
            return hours*60 + mins if (hours or mins) else np.nan
        nums = [int(tok) for tok in s.split() if tok.isdigit()]
        return nums[0] if nums else np.nan
    except: return np.nan

df_use['Duration'] = df_use['Duration'].apply(parse_duration_to_min)
df_use['Duration'].fillna(df_use['Duration'].median(), inplace=True)

def split_to_list(cell):
    if pd.isna(cell) or str(cell).strip()=="": return []
    return [entry.strip() for entry in str(cell).replace(';',',').split(',') if ent

for col in ['Genres','Producers','Licensors','Studios']:
    df_use[col] = df_use[col].apply(split_to_list)

df_use['Source'] = df_use['Source'].fillna("Unknown").astype(str)
df_use['Rating'] = df_use['Rating'].fillna("Unknown").astype(str)

print("Feature preprocessing done.")
print("Sample data:\n", df_use.head(2))
```

```
Feature preprocessing done.
Sample data:
    MAL_ID                              Name  \
0       1                      Cowboy Bebop
1       5  Cowboy Bebop: Tengoku no Tobira

                                          Genres            English name  \
0  [Action, Adventure, Comedy, Drama, Sci-Fi, Space]            Cowboy Bebop
1            [Action, Drama, Mystery, Sci-Fi, Space]  Cowboy Bebop:The Movie

    Japanese name   Type  Episodes                         Aired    Premiered  \
0       カウボーイビバップ     TV      26.0  Apr 3, 1998 to Apr 24, 1999  Spring 1998
1  カウボーイビバップ 天国の扉  Movie       1.0                 Sep 1, 2001      Unkn
own

                  Producers  ... Ranked Popularity  Members  Favorites  \
0           [Bandai Visual]  ...   28.0         39  1251960      61971
1  [Sunrise, Bandai Visual]  ...  159.0        518   273145       1174

  Watching Completed  On-Hold  Dropped  Plan to Watch  Score_Class
0   105808    718161    71513    26678         329800    Very Good
1     4143    208333     1935      770          57964    Very Good

[2 rows x 25 columns]
C:\Users\samasup\AppData\Local\Temp\ipykernel_6772\224860240.py:4: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df_use['Episodes'].fillna(df_use['Episodes'].median(), inplace=True)
C:\Users\samasup\AppData\Local\Temp\ipykernel_6772\224860240.py:22: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df_use['Duration'].fillna(df_use['Duration'].median(), inplace=True)
```

In [19]:

```
min_genre_count = 30
mlb_gen = MultiLabelBinarizer()
genres_matrix = pd.DataFrame(mlb_gen.fit_transform(df_use['Genres']), columns=mlb_g
keep_genres = genres_matrix.columns[genres_matrix.sum(axis=0) >= min_genre_count]
genres_matrix = genres_matrix[keep_genres]

def top_k_multi_label(df_series, k):
```

```
    all_items = Counter()
    for lst in df_series: all_items.update(lst)
    return [item for item, _ in all_items.most_common(k)]

def make_topk_binary(df_series, topk):
    return pd.DataFrame({item: df_series.apply(lambda lst, it=item: int(it in lst))

producers_matrix = make_topk_binary(df_use['Producers'], top_k_multi_label(df_use['
studios_matrix = make_topk_binary(df_use['Studios'], top_k_multi_label(df_use['Stud
licensors_matrix = make_topk_binary(df_use['Licensors'], top_k_multi_label(df_use['

X = pd.concat([
    df_use[['Episodes','Duration','Source','Rating']].reset_index(drop=True),
    genres_matrix.reset_index(drop=True),
    producers_matrix.reset_index(drop=True),
    studios_matrix.reset_index(drop=True),
    licensors_matrix.reset_index(drop=True)
], axis=1)

y = df['Score_Class'].loc[X.index].copy()
print("Final feature matrix shape:", X.shape)
print("Classes in target:", y.unique())
```

```
Final feature matrix shape: (12421, 147)
Classes in target: ['Very Good' 'Good' 'Average' 'Low']
```

In [20]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print("Train size:", len(X_train), "Test size:", len(X_test))

numeric_cols = ['Episodes', 'Duration']
categorical_cols = ['Source', 'Rating']

enc_params = {"handle_unknown": "ignore"}
if sklearn.__version__ >= "1.2":
    enc_params["sparse_output"] = False
else:
    enc_params["sparse"] = False

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(**enc_params), categorical_cols)
], remainder='passthrough')

print("Preprocessor setup done.")
```

```
Train size: 9936 Test size: 2485
Preprocessor setup done.
```

In [21]:

```
svm_clf = SVC(kernel='rbf', C=10.0, gamma='scale', class_weight='balanced', random_
pipeline = Pipeline([
    ('preproc', preprocessor),
    ('clf', svm_clf)
])
pipeline.fit(X_train, y_train)
print("SVM training completed.")
```

```
SVM training completed.
```

In [22]:
```
y_pred = pipeline.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)

print("\nConfusion Matrix:")
print(pd.DataFrame(confusion_matrix(y_test, y_pred, labels=['Average','Good','Low',
                   index=['Average','Good','Low','Very Good'],
                   columns=['Average','Good','Low','Very Good']))

print("\nClassification Report:")
print(classification_report(y_test, y_pred, labels=['Average','Good','Low','Very Go

print("\nTest class counts:\n", y_test.value_counts())
```

```
Accuracy: 0.6354124748490946

Confusion Matrix:
           Average  Good  Low  Very Good
Average        667   172  212          9
Good           165   408   35         39
Low            182    26  460          0
Very Good        4    60    2         44

Classification Report:
              precision    recall  f1-score   support

     Average       0.66      0.63      0.64      1060
        Good       0.61      0.63      0.62       647
         Low       0.65      0.69      0.67       668
   Very Good       0.48      0.40      0.44       110

    accuracy                           0.64      2485
   macro avg       0.60      0.59      0.59      2485
weighted avg       0.63      0.64      0.63      2485


Test class counts:
 Score_Class
Average      1060
Low           668
Good          647
Very Good     110
Name: count, dtype: int64
```

Summary & Observations

- Accuracy: ~63.5%
- Best performance for Average and Low classes.
- Very Good class has few samples, leading to lower recall (0.40).
- Multi-label features like genres, producers, and studios are important.
- Future Work:
  - Hyperparameter tuning

- Try ensemble classifiers
- Reduce dimensionality (PCA/feature selection)
- Use embeddings for multi-label features