Support Vector Machine (SVM) Report

Section 1

# APR Assignment: Anime Score Classification

Name: Nenavath Suresh
Roll No: 2201AI25

Section 2

## 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.

Section 3

## 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

Appendix: Code Snippets

Code Cell 1

```
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, StandardScaler
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__)
```

Code Cell 2

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

Code Cell 3

```
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 ==
'Score']
df_features = df.drop(columns=score_cols)

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

Code Cell 4

```
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("hr"): hours =
int(token)
if token.isdigit() and i+1 < len(parts) and parts[i+1].startswith("min"): mins =
int(token)
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
entry.strip()]

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))
```

Code Cell 5

```
min_genre_count = 30
mlb_gen = MultiLabelBinarizer()
genres_matrix = pd.DataFrame(mlb_gen.fit_transform(df_use['Genres']),
columns=mlb_gen.classes_, index=df_use.index)
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)) for
item in topk})

producers_matrix = make_topk_binary(df_use['Producers'],
top_k_multi_label(df_use['Producers'], 40))
studios_matrix = make_topk_binary(df_use['Studios'],
top_k_multi_label(df_use['Studios'], 40))
licensors_matrix = make_topk_binary(df_use['Licensors'],
top_k_multi_label(df_use['Licensors'], 20))

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())
```

Code Cell 6

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)
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.")
```

Code Cell 7

```
svm_clf = SVC(kernel='rbf', C=10.0, gamma='scale', class_weight='balanced',
random_state=42)
pipeline = Pipeline([
('preproc', preprocessor),
('clf', svm_clf)
])
pipeline.fit(X_train, y_train)
print("SVM training completed.")
```

Code Cell 8

```
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','Very Good']),
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
Good']))

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