Discovering and Classifying Film Genres Using Unsupervised and Supervised Learning

Toma Sabin Sebastian – Group 412

Introduction

In my research, I utilized a large dataset of films dating back to 1907. This dataset includes basic information such as release dates and directors, as well as detailed data like cast lists, titles, production details, and in-depth plot summaries. Analyzing this information can help uncover meaningful patterns related to film genres and thematic similarities. In this project, I aim to explore the use of both unsupervised clustering and supervised learning techniques to better understand how to group the films and predict their genres. My focus is on two main unsupervised approaches: *Birch Clustering* – leveraging text-based features(TF-IDF and Word2Vec embeddings generated from film plots) alongside other film attributes to predict the genre of the film and *OPTICS Clustering* - applying a density-based approach to cluster films using primarily numeric and categorical metadata, without relying on plot features. However, to evaluate how well unsupervised methods perform in predicting an actual labeled outcome(the film's genre) or grouping movies after distance to centroid, I compare against: Supervised baselines – such as *RandomForest and XGBoost* trained on labeled data and Random chance – a basic baseline that serves as a lower bound. Additionally, I included an extra *K-Means* clustering experiment to see how a more traditional centroid-based algorithm compared to OPTICS. By testing these various on the same dataset, I can assess their relative performance and limitations in both clutering methods and prediction tasks. In the process, I tune relevant hyperparameters, like the distance thresholds in OPTICS and the branching factor in Birch to optimize results. Ultimately, the project demonstrates how different types of features(text vs numeric) and different models(supervised vs unsupervised) can complement each other to yield valuable insights into film data.

Dataset description and preprocessing

Dataset overview

The dataset used for this project contains detailed metadata about films from 1907 onward, including basic attributes such as *release year, title, and director,* as well as rich information like *cast lists, production details, plot summaries, and genres*. The dataset was thoroughly cleaned to ensure reliability and consistency by removing rows with missing or unknown values in key columns, such as *Plot*, *Genre*, *Director*, and *Cast*. After filtering and preprocessing, the cleaned dataset was saved and utilized for subsequent clustering and classification tasks. The final dataset was further filtered to include only genres with at least 900 occurrences to ensure sufficient representation for meaningful analysis, resulting in a balanced and high-quality dataset.

Data splitting for supervised learning models

The dataset was split into training, validation and testing subsets: training set – 80%, used for model training, validation set – 10%, used for parameter tuning and model evaluation and testing set – 10%, used exclusively for final evaluation. This splitting ensured a clear separation of data for training and evaluation, adhering to best practices to prevent data leakage.

Feature engineering

Categorical metadata features, such as *Origin/Ethnicity* and *Genre*, were encoded using *Label Encoding* to convert textual labels into numeric representations. *Origin/Ethnicity*: Transformed into integer values representing unique origins. *Genre*: Encoded into numeric labels to serve as ground truth for supervised models and for post-hoc evaluation of clustering methods. Numeric features, such as *Release Year*, were scaled using *StandardScaler* to ensure all features contributed equally to distance-based clustering methods. The *Plot* column was processed using the *Term Frequency-Inverse Document Frequency (TF-IDF)* technique to create a sparse matrix of features: The top 100 most significant words were extracted using TF-IDF vectorization to capture the importance of words in each plot relative to the entire dataset, emphasizing unique and meaningful terms for clustering. To capture semantic relationships within the *Plot* column,

Word2Vec embedding was used where plots were tokenized into sentences and each word was embedded into a 100-dimensional vector space and for each film, the *Word2Vec embeddings* of all words in its plot were averaged to create a fixed-length vector. *Word2Vec* was also used to generate embeddings for Title and Cast columns. *Unsupervised Clustering**:*** The *Genre* labels were never used during the training or hyperparameter tuning of unsupervised methods (Birch and OPTICS). The labels were used only for post-hoc evaluation to assess clustering quality through metrics like Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). *Supervised Models*: For Random Forest and XGBoost, labels were used during training. These models were evaluated on the test set, ensuring a clear separation between training and evaluation phases.

Birch Clustering

Features used: Combined TF-IDF and Word2Vec embeddings for the *Plot* and *Title* columns, along with numerical metadata (*Release Year*, *Origin/Ethnicity*).

Preprocessing: Features were scaled using StandardScaler to ensure equal contribution. Dimensionality was reduced with PCA to improve efficiency and reduce noise. Oversampling with RandomOverSampler balanced the genre distribution before clustering.

```
# Preprocessing
# Drop irrelevant columns
data = data.drop(columns=['Wiki Page', 'Director', 'Cast'])

# Encode categorical variables
label_encoder = LabelEncoder()
data['Origin/Ethnicity'] = label_encoder.fit_transform(data['Origin/Ethnicity'])
data['Genre'] = label_encoder.fit_transform(data['Genre'])

✓ 0.0s
```

Figure 1. Preprocessing

I removed the columns that are not relevant for clustering or feature extraction(Wiki *Page*: an url that is not useful for clustering; *Director and Cast*:Not directly using in clustering due to insufficient contextual information). *Origin/Ethnicity* is encoded into integers, allowing it

to be used as a numeric feature and *Genre* is encoded into integers for supervised evaluation and balancing the dataset but NOT directly used during clustering.

```
# TF-IDF for 'Plot' column
tfidf_vectorizer = TfidfVectorizer(max_features=100)
plot_tfidf = tfidf_vectorizer.fit_transform(data['Plot'].fillna(''))
```
✓ 4.3s

Figure 2. TF-IDF for Plot

Converts the *Plot* column into a sparse matrix of features using TF-IDF. Only the top 100 words (most statistically significant) are retained and captures the importance of individual words within the plot relative to the entire dataset.

```
# Word2Vec for 'Plot' column
tokenized_plots = data['Plot'].fillna('').apply(lambda x: x.split())
word2vec_model = Word2Vec(sentences=tokenized_plots, vector_size=80, window=5, min_count=1, workers=4)
```
✓ 29.6s

```
def get_plot_embedding(plot):
    embeddings = [word2vec_model.wv[word] for word in plot if word in word2vec_model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(word2vec_model.vector_size)

plot_word2vec = tokenized_plots.apply(get_plot_embedding)
plot_word2vec = np.vstack(plot_word2vec.values)
```

Figure 3. Word2Vec for Plot

Tokenizes the *Plot* into words and trains a Word2Vec model. Each word is embedded into an 80-dimensional vector space. Then, it averages the Word2Vec embeddings for all words in each plot, resulting in a fixed-length 80-dimensional vector for each film.

```python
# TF-IDF for 'Title' column
title_tfidf_vectorizer = TfidfVectorizer(max_features=50)
title_tfidf = title_tfidf_vectorizer.fit_transform(data['Title'].fillna(''))

# Word2Vec for 'Title' column
tokenized_titles = data['Title'].fillna('').apply(lambda x: x.split())
title_word2vec_model = Word2Vec(sentences=tokenized_titles, vector_size=50, window=3, min_count=1, workers=4)

def get_title_embedding(title):
    embeddings = [title_word2vec_model.wv[word] for word in title if word in title_word2vec_model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(title_word2vec_model.vector_size)

title_word2vec = tokenized_titles.apply(get_title_embedding)
title_word2vec = np.vstack(title_word2vec.values)
```
✓ 0.6s

Figure 4. TF-IDF & Word2Vec for Title

Similar preprocessing is applied to the *Title* column, with TF-IDF generating 50 features and Word2Vec creating 50-dimensional embeddings.

```python
# Combine TF-IDF and Word2Vec features for 'Plot' and 'Title'
combined_text_features = np.hstack((plot_word2vec, plot_tfidf.toarray(), title_word2vec, title_tfidf.toarray()))
```
✓ 0.0s

```python
# Combine numeric and textual features
numeric_features = data[['Release Year', 'Origin/Ethnicity', 'Genre']].values
numeric_features = StandardScaler().fit_transform(numeric_features)
combined_features = np.hstack((numeric_features, combined_text_features))
```
✓ 0.0s

Figure 5. Combined features

Text-based features combine Word2Vec and TF-IDF features for *Plot* and *Title*. Numeric features include *Release Year*, *Origin/Ethnicity* and *Genre* (though *Genre* is used only for oversampling and evaluation, not clustering). StandardScaler ensures all features have equal weight by scaling them to zero mean and unit variance.

```python
# Apply RandomOverSampler to balance the data
oversampler = RandomOverSampler(random_state=42)
combined_features_oversampled, labels_oversampled = oversampler.fit_resample(combined_features, data['Genre'])
```
✓ 0.0s

Figure 6. Balancing the dataset

RandomOverSampler balances the dataset by oversampling minority genres. This ensures all genres are equally represented in the data fed to the clustering algorithm.

```
# Apply Birch Clustering on the oversampled data
birch_model = Birch(n_clusters=7, threshold=0.3, branching_factor=50)
birch_model.fit(combined_features_oversampled)

# Get cluster labels for the oversampled data
oversampled_cluster_labels = birch_model.labels_
```

✓ 1m 14.6s

Figure 7. Birch clustering

- n_clusters: number of clusters
- threshold: radius for merging sub-clusters
- branching-factor: maximum number of child nodes in the clustering features(CF) tree.

```
# Map cluster labels to true labels
mapped_labels = map_labels(labels_oversampled, birch_model.labels_)

# Calculate cluster accuracy
cluster_accuracy = accuracy_score(labels_oversampled, mapped_labels)
print(f"Cluster Accuracy: {cluster_accuracy}")

# Evaluate clustering performance
# Silhouette Score
silhouette_avg = silhouette_score(combined_features_oversampled, birch_model.labels_)
print(f"Silhouette Score: {silhouette_avg}")

# Adjusted Rand Index (ARI)
ari_score = adjusted_rand_score(labels_oversampled, birch_model.labels_)
print(f"Adjusted Rand Index (ARI): {ari_score}")

# Normalized Mutual Information (NMI)
nmi_score = normalized_mutual_info_score(labels_oversampled, birch_model.labels_)
print(f"Normalized Mutual Information (NMI): {nmi_score}")
```

✓ 6m 25.6s

```
Cluster Accuracy: 0.5414883296239228
Silhouette Score: 0.08590233019112331
Adjusted Rand Index (ARI): 0.36763908135601525
Normalized Mutual Information (NMI): 0.5057961324416321
```

Figure 8. Cluster evaluation

Cluster accuracy: This indicates that 54.15% of the samples were correctly assigned to their respective clusters after mapping the cluster labels to true labels. While this accuracy suggests that the clustering captures meaningful relationships between films and their genres, it also highlights that nearly half of the samples were misclassified, pointing to overlaps or ambiguities in the data.

Silhouette score: A Silhouette Score close to 0 implies that the clusters are either overlapping or poorly separated. In this case, the low score reflects the challenge of clustering genres, as many genres (comedy and drama, or thriller and action) may share common themes and vocabulary in their titles or plots, leading to blurred cluster boundaries.

Adjusted Rand Index: The ARI of 0.368 indicates moderate agreement between the predicted clusters and the true genre labels. This score shows that while the clustering model captures some meaningful structure in the data, it struggles to perfectly align clusters with the true genres, likely due to significant overlap in the feature space of certain genres.

Normalized Mutual Information: An NMI score of 0.506 suggests that the information shared between the clustering assignments and the true labels is moderate. This reflects that the model has successfully captured some genre-specific patterns but is limited in fully distinguishing between genres with overlapping features.

The PCA scatter plot shows a 2D projection of the clusters (post-oversampling). The clusters are moderately well-separated, but some overlaps are visible, suggesting similarities between certain genres.

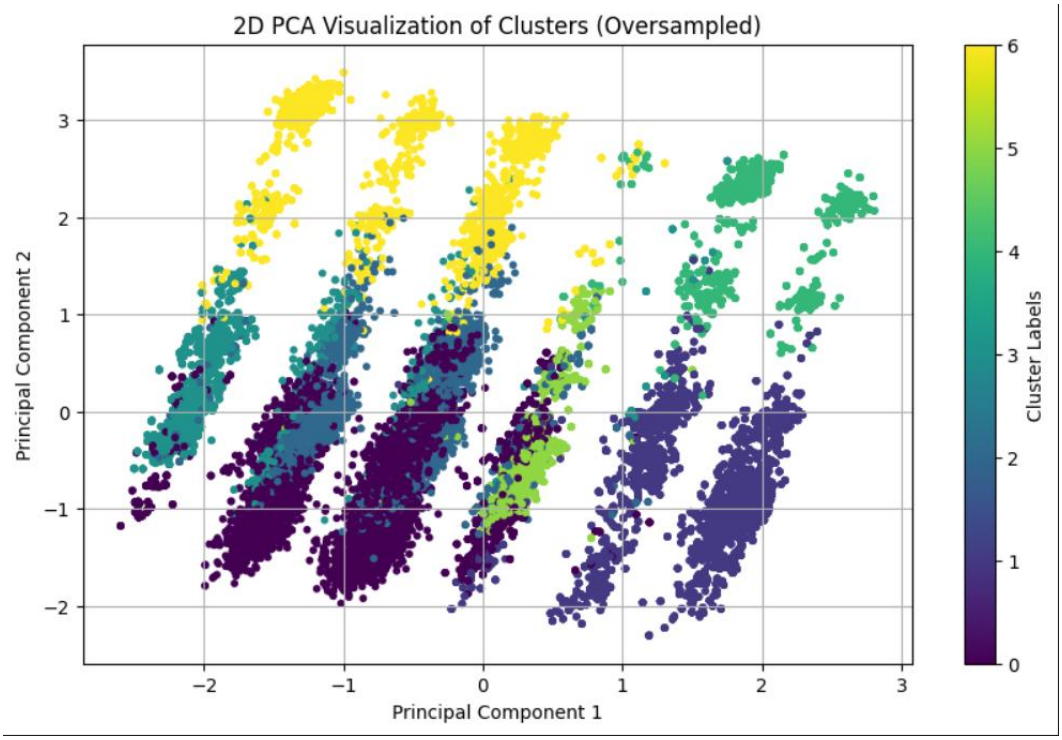

Figure 9. Visualization of clusters with TF-IDF and Word2Vec embeddings

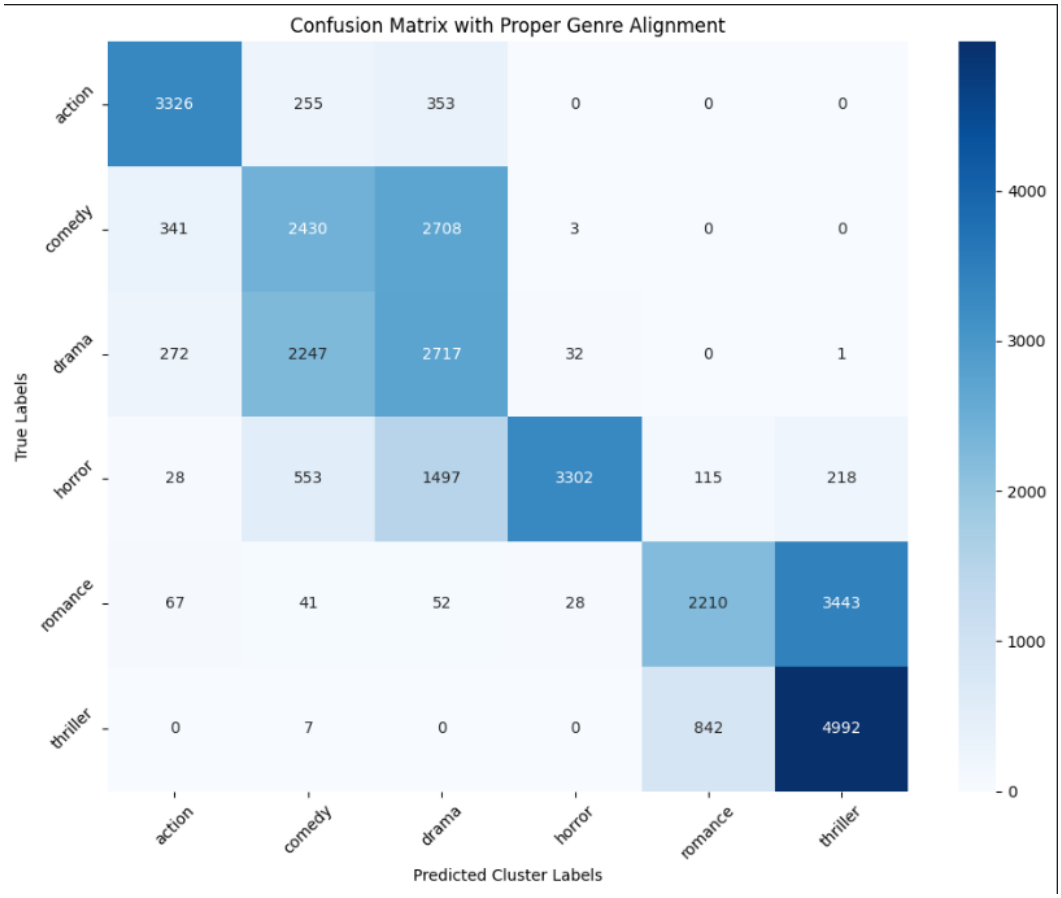

Figure 10. Confusion matrix with TF-IDF and Word2Vec embeddings

The confusion matrix illustrates the relationship between true labels (genres) and predicted cluster labels.

- Action(Cluster 0): The cluster contains a high number of correctly identified "action" films. The model's embeddings successfully captured distinguishing features, likely from title and plot keywords, such as "fight," "battle," or "hero."
- Comedy & Drama(Clusters 1 & 2): Significant confusion between these two genres, reflected in a high number of comedy films being clustered with drama. This overlap is expected since comedy films often blend dramatic elements, and plot summaries might not always highlight comedic aspects.
- Horror(Cluster 3): The horror genre forms a relatively distinct cluster with moderate purity. Typical horror-specific vocabulary and patterns ("ghost," "haunted," "fear") likely contributed to its clear separation.
- Romance & Thriller(Clusters 4 & 5): Romance and thriller genres are fairly well-separated but still show some confusion, especially with thriller films misclassified as romance. Shared narrative elements like emotional intensity or suspense could account for the overlap.

Attempt with other hyperparameters:

```
# Word2Vec for 'Plot' column
tokenized_plots = data['Plot'].fillna('').apply(lambda x: x.split())
word2vec_model = Word2Vec(sentences=tokenized_plots, vector_size=100, window=5, min_count=1, workers=4)

# Apply Birch Clustering on the oversampled data
birch_model = Birch(n_clusters=6, threshold=0.3, branching_factor=30)
birch_model.fit(combined_features_oversampled)
```

```
Cluster Accuracy: 0.39887576328254293
Silhouette Score: 0.14760313925614 42
Adjusted Rand Index (ARI): 0.24148799908467372
Normalized Mutual Information (NMI): 0.38044792820139106
```

Figure 11. Results with other hyperparameters

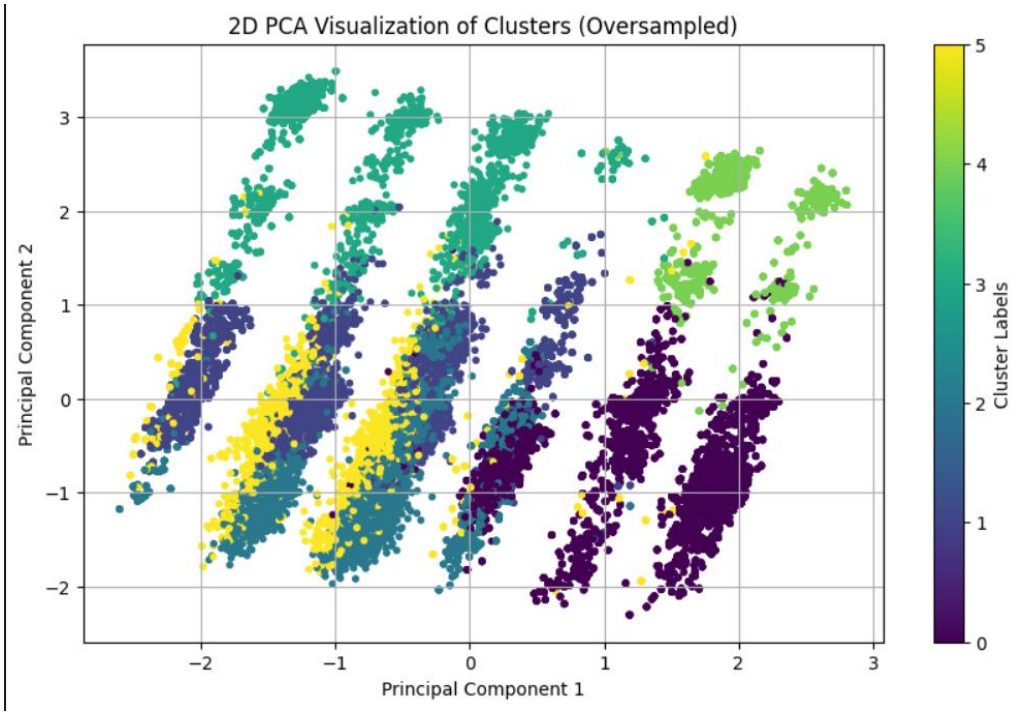

Figure 12. Visualization of clusters with other parameters

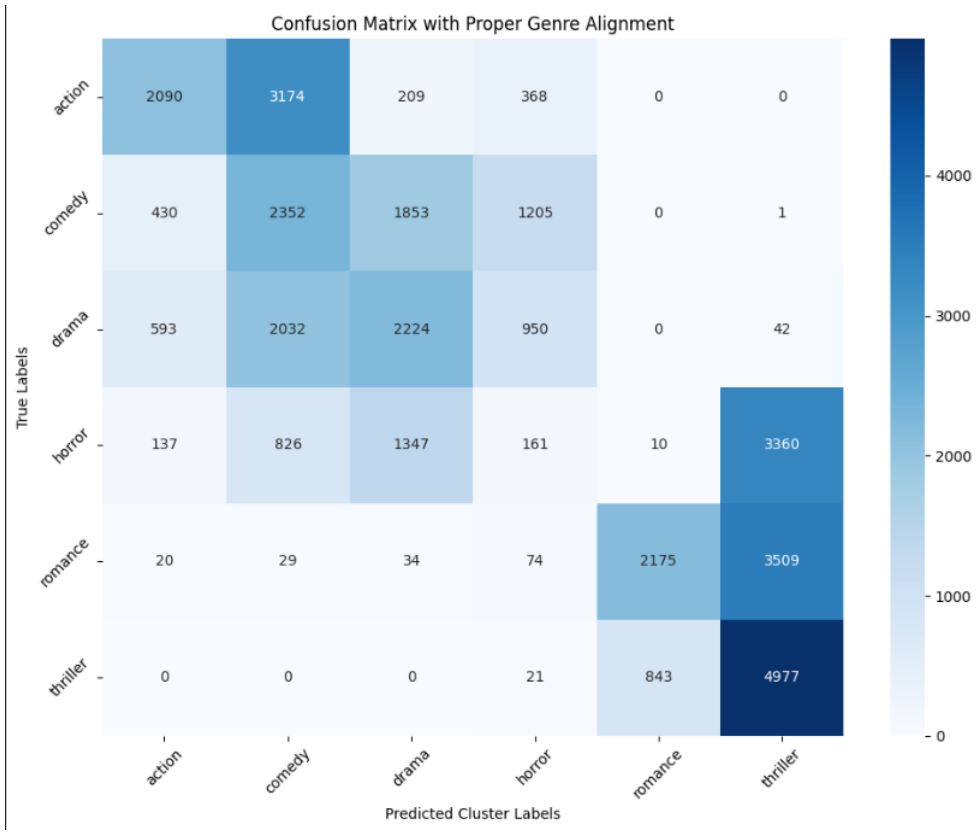

Figure 13. Confusion matrix with other hyperparameters

Attempt with Birch & TF-IDF: In this attempt, the same parameters and hyperparameters were used as in the default model (the first one I discussed). The only difference is that **Word2Vec embeddings were excluded** and only TF-IDF features were used.

```
Cluster Accuracy: 0.5377798507462687
Silhouette Score: 0.293371168446290 35
Adjusted Rand Index (ARI): 0.3690622089570313
Normalized Mutual Information (NMI): 0.5357080930337832
```

Figure 14. Results with Birch & TF-IDF

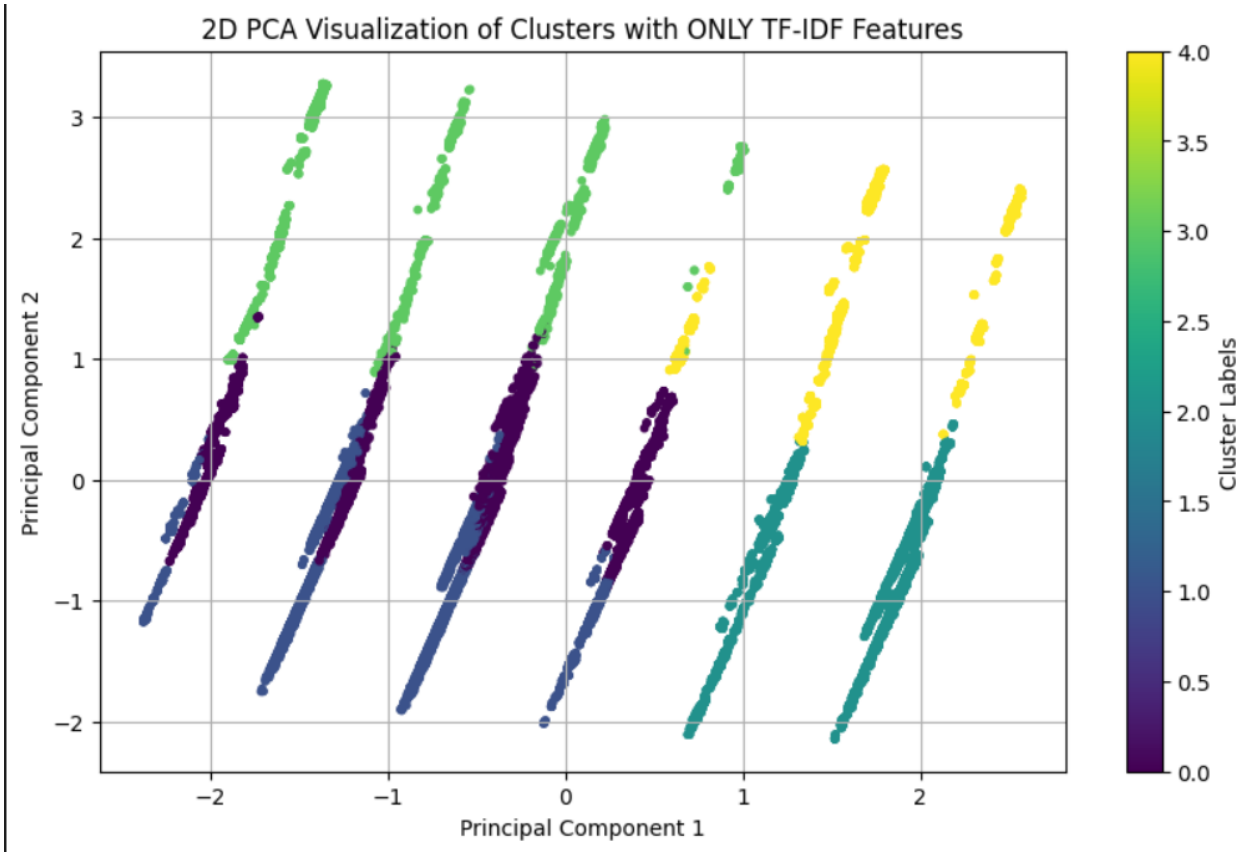

Figure 15. Visualization of clusters with Birch & TF-IDF

Attempt with Birch & Word2vec: In this attempt, the **TD-IDF embedding were excluded** and the parameters are:

```
# Train Word2Vec model on the tokenized plots
word2vec_model = Word2Vec(sentences=tokenized_plots, vector_size=100, window=5, min_count=1, workers=4)

# Apply Birch Clustering on the oversampled data
birch_model = Birch(n_clusters=5, threshold=0.2, branching_factor=30)
birch_model.fit(combined_features_oversampled)
```

```
Cluster Accuracy: 0.451308139534883 75
Silhouette Score: 0.16240132237892185
Adjusted Rand Index (ARI): 0.29736106452532557
Normalized Mutual Information (NMI): 0.4094525638790598
```

Figure 16. Results with Birch & Word2Vec

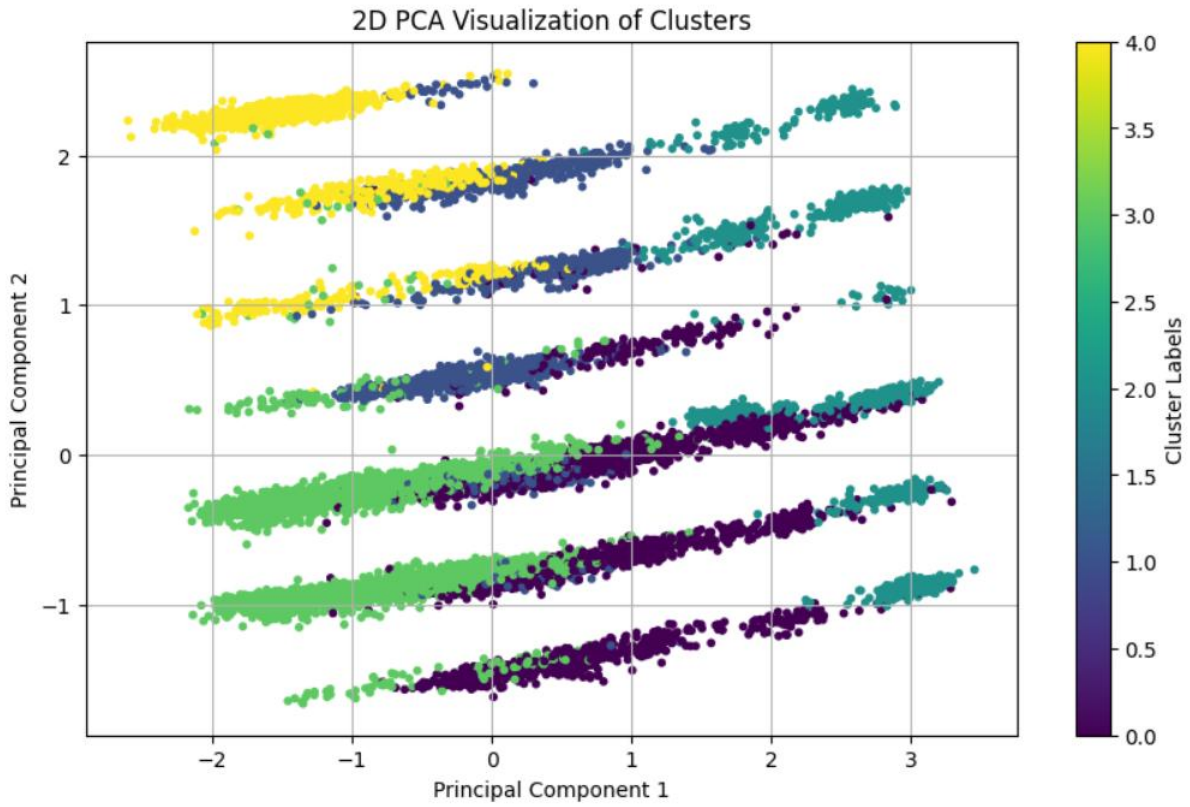

Figure 17. Visualization of clusters with Birch & Word2Vec

Supervised attempt: Random Forest Classification with Grid Search and Smote

The goal of this supervised learning attempt was to predict the *Genre* of films using a combination of numeric metadata (*Release Year*, *Origin/Ethnicity*) and text-based features (*Plot* and *Title*). A *Random Forest Classifier* was trained and optimized using *Grid Search* to select the best hyperparameters, and the training data was balanced using *SMOTE* to handle genre imbalances, for the *Plot and Title* embeddings were used TF-IDF and Word2Vec.

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Initialize the Random Forest classifier
rf_model = RandomForestClassifier(random_state=42)

# Perform grid search with the SMOTE-resampled data
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, scoring='accuracy', cv=3, verbose=1, n_jobs=-1)
grid_search.fit(X_train_smote, y_train_smote)
```

Figure 18. Smote, RandomForest & GridSearch

```
Best Parameters from Grid Search: {'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Best Cross-Validation Accuracy: 0.82
Validation Accuracy: 0.53
Test Accuracy: 0.52
```

Figure 19. Results with supervised attempt

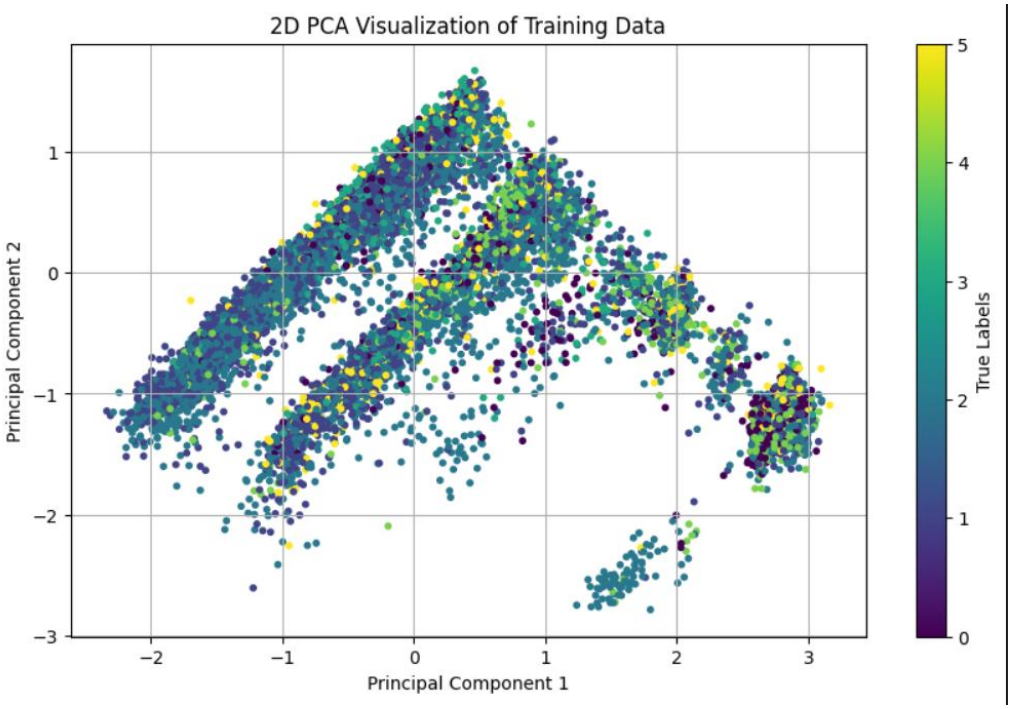

Figure 20. Visualization of Training Data on supervised attempt

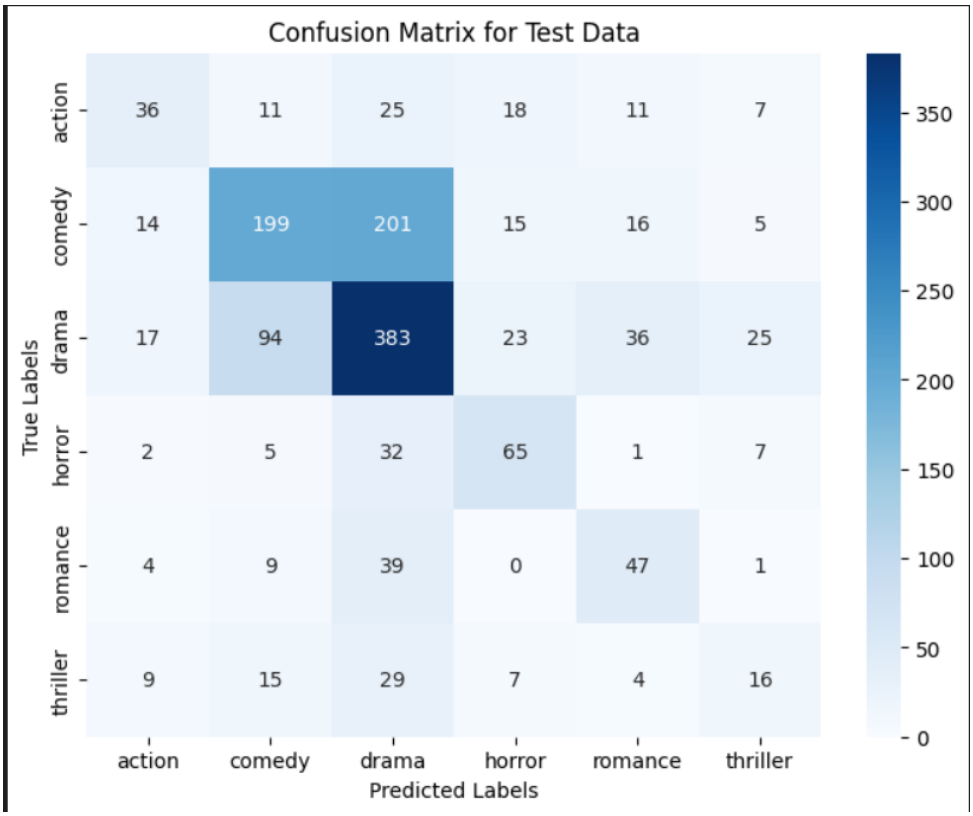

Figure 21. Confusion matrix on supervised attempt

Random chance attempt: Cluster assignment and evaluation

This code simulates the performance of random cluster assignment as a baseline for evaluating clustering algorithms. The goal is to provide a comparison point for meaningful clustering results by assigning random cluster labels to the oversampled data and calculating key evaluation metrics. It was used TF-IDF and Word2Vec for *Plot and Title* columns and RandomOverSampler balances the dataset by oversampling underrepresented genres, ensuring all genres are equally represented during evaluation.

```
Cluster Accuracy: 0.1721737145465959
Silhouette Score: -0.0033653293589463507
Adjusted Rand Index (ARI): 2.9365778529118l7e-05
Normalized Mutual Information (NMI): 0.0002400179849633628 5
```

Figure 22. Evaluation metrics on random chance attempt

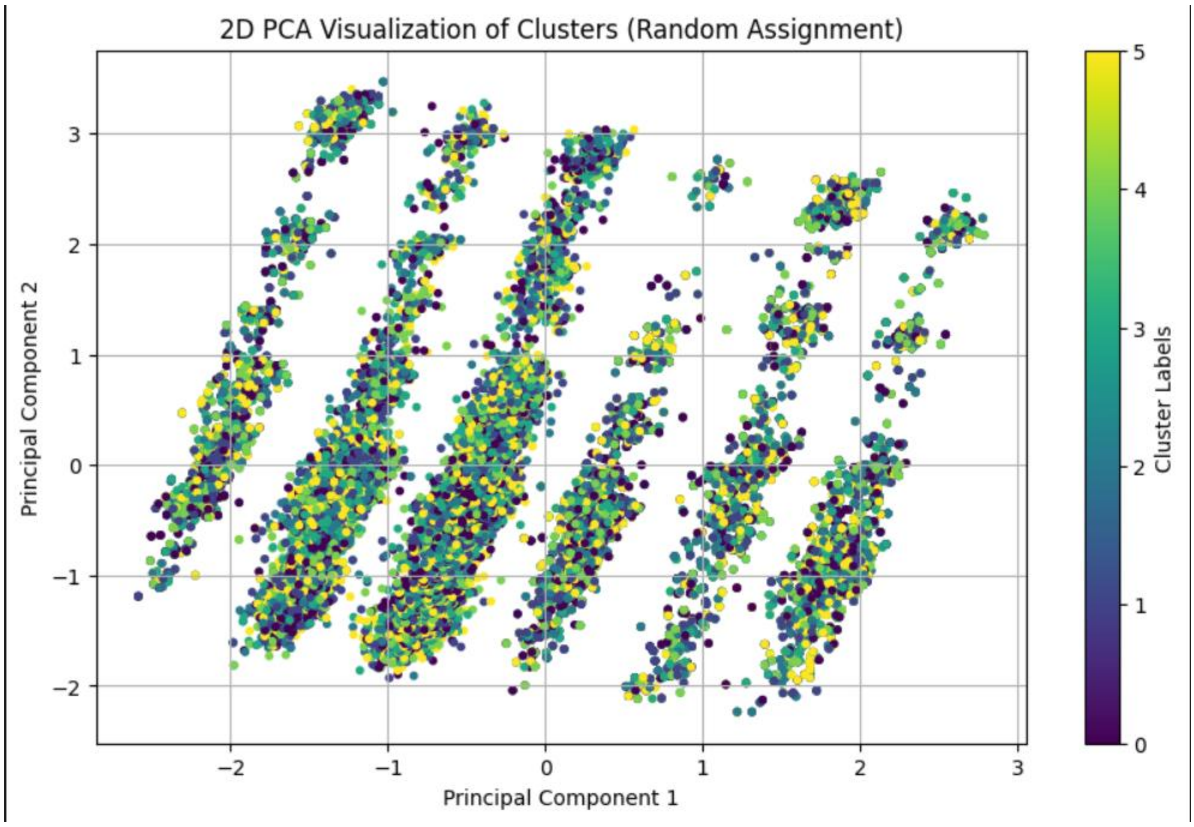

Figure 23. Visualization of clusters on random chance attempt

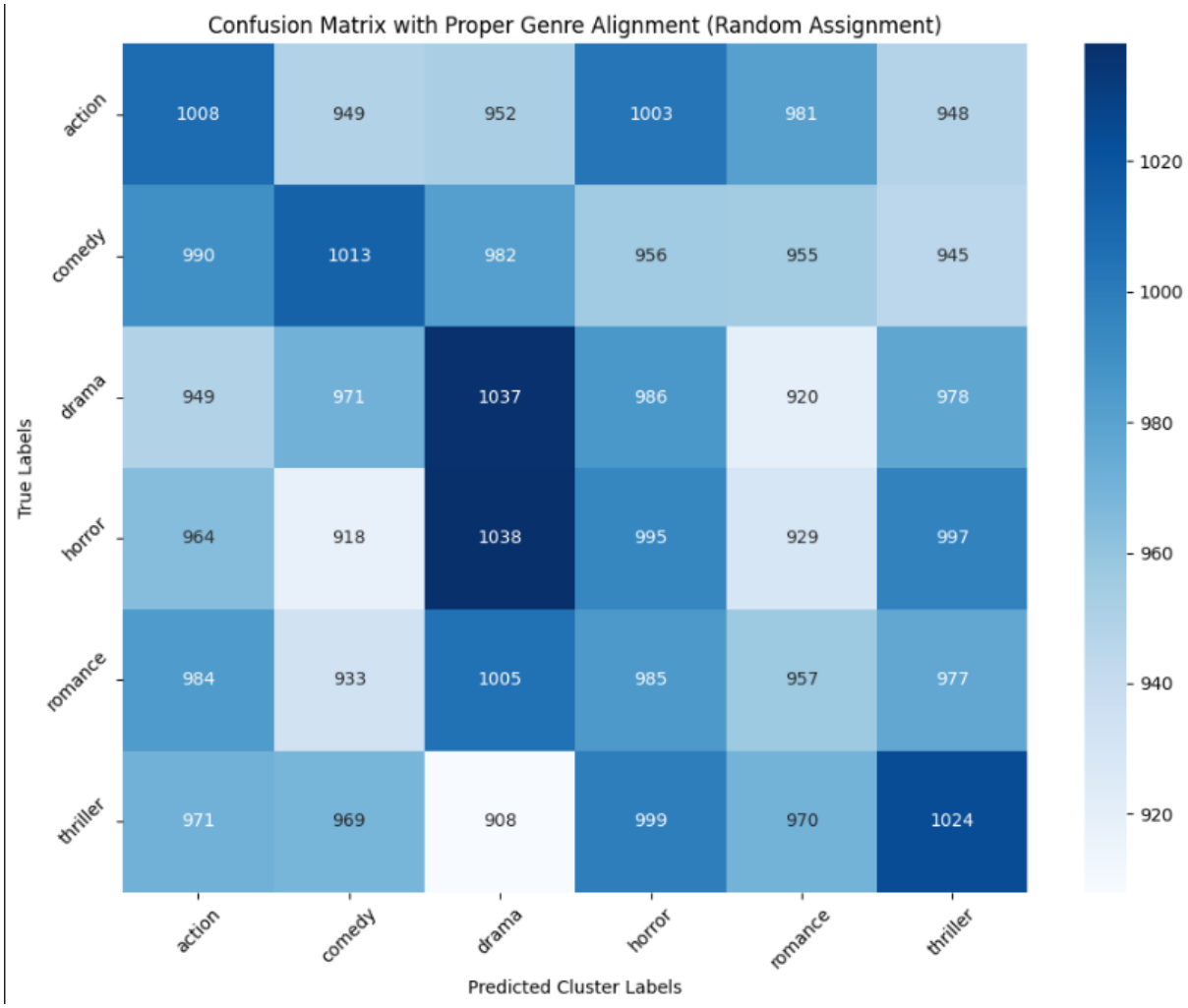

Figure 24. Confusion matrix on random chance attempt

Conclusions

Supervised learning achieves higher accuracy compared to clustering but depends heavily on labeled data. The performance gap between cross-validation and test accuracy highlights the need for techniques to improve generalization.

Birch clustering provides moderate alignment with true genres, making it a useful tool for exploratory analysis. Clustering relies on the quality of feature representation, improving textual features could enhance performance.

Random clustering serves as a baseline to demonstrate the effectiveness of meaningful clustering. The poor performance across all metrics shows the necessity of clustering approaches that leverage the structure in data.

Optics clustering

Features used: Original numeric features – *Realease Year and Origin/Ethnicity*(encoded as a numeric feature using *LabelEncoder*). I generated embeddings using Word2Vec for textual columns – *Title and Cast embeddings*. Also, for both *Title and Cast embeddings* , I extracted *maximum value, variance and standard deviation* resulting in 6 statistical features. Finally, after processing, I combined the features having 208 features.

Prepocessing: I handled the missing data, I encoded categorical variables, tokenized the text columns, generated Word2Vec embeddings from *Title and Cast* columns, added statistical features, combined all features and scaled them. After that, I used SMOTE to balance the dataset, generating synthethic samples for minority classes in the *Genre* column to ensure all classes have similar representation.

```
# Tokenize 'Title' and 'Cast' for Word2Vec
data['Title_Tokens'] = data['Title'].fillna('').apply(lambda x: x.split())
data['Cast_Tokens'] = data['Cast'].fillna('').apply(lambda x: x.split(','))
```
✓ 0.0s

```
# Train Word2Vec for 'Title' and 'Cast'
title_w2v_model = Word2Vec(sentences=data['Title_Tokens'], vector_size=100, window=3, min_count=1, workers=4)
cast_w2v_model = Word2Vec(sentences=data['Cast_Tokens'], vector_size=100, window=3, min_count=1, workers=4)
```
✓ 0.8s

```
# Function to get Word2Vec embeddings
def get_w2v_embedding(tokens, model, vector_size):
    embeddings = [model.wv[token] for token in tokens if token in model.wv]
    if embeddings:
        return np.mean(embeddings, axis=0)
    else:
        return np.zeros(vector_size)
```
✓ 0.0s

```
# Generate embeddings for 'Title' and 'Cast'
data['Title_Embedding'] = data['Title_Tokens'].apply(lambda x: get_w2v_embedding(x, title_w2v_model, 10))
data['Cast_Embedding'] = data['Cast_Tokens'].apply(lambda x: get_w2v_embedding(x, cast_w2v_model, 10))
```
✓ 0.5s

```
# Expand embedding columns into separate features
title_embeddings = np.vstack(data['Title_Embedding'].values)
cast_embeddings = np.vstack(data['Cast_Embedding'].values)
```
✓ 0.0s

Figure 25. Title & Cast embeddings

After that, I removed the columns that I will not use in clustering.

```
# Combine all features
numeric_features = data[['Release Year', 'Origin/Ethnicity']].values
combined_features = np.hstack((numeric_features, title_embeddings, cast_embeddings))
```

✓ 0.0s

Generate + Code + Markdow

```
# Add refined statistical features
max_title_embedding = np.max(title_embeddings, axis=1).reshape(-1, 1)
max_cast_embedding = np.max(cast_embeddings, axis=1).reshape(-1, 1)
var_title_embedding = np.var(title_embeddings, axis=1).reshape(-1, 1)
var_cast_embedding = np.var(cast_embeddings, axis=1).reshape(-1, 1)
std_title_embedding = np.std(title_embeddings, axis=1).reshape(-1, 1)
std_cast_embedding = np.std(cast_embeddings, axis=1).reshape(-1, 1)
```

✓ 0.0s

```
# Add features to the combined features
combined_features = np.hstack((
    combined_features,
    max_title_embedding,
    max_cast_embedding,
    var_title_embedding,
    var_cast_embedding,
    std_title_embedding,
    std_cast_embedding
))
```

Figure 26. Features used in Optics model

```
# Standardize all features
scaler = StandardScaler()
data_scaled = scaler.fit_transform(combined_features)
```

✓ 0.0s

Figure 27. Standardization

```
# Apply SMOTE
true_labels = data['Genre']
smote = SMOTE(random_state=42)
data_balanced, true_labels_balanced = smote.fit_resample(data_scaled, true_labels)
```

✓ 0.4s

```
print("Data shape after SMOTE:", data_balanced.shape)
print("Class distribution after SMOTE:", pd.Series(true_labels_balanced).value_counts())
```

✓ 0.0s

```
Data shape after SMOTE: (35046, 208)
Class distribution after SMOTE: Genre
comedy      5841
drama       5841
horror      5841
romance     5841
thriller    5841
action      5841
Name: count, dtype: int64
```

Figure 28. Balancing the dataset

```
# Reduce to 10 principal components
pca = PCA(n_components=10)
data_reduced = pca.fit_transform(data_balanced)
```

✓ 0.1s

Generate

```
# Apply OPTICS clustering
optics_model = OPTICS(min_samples=5, xi=0.02, min_cluster_size=0.01)
optics_model.fit(data_reduced)
```

✓ 1m 44.7s

OPTICS

```
OPTICS(min_cluster_size=0.01, xi=0.02)
```

```
# Get cluster labels
cluster_labels = optics_model.labels_
```

✓ 0.0s

Figure 29. Optics model

- min_samples: The minimum number of data points required to form a dense region (core point).
- xi: A factor to determine the minimum relative decrease in the reachability distance required to form a new cluster.
- min_cluster_size: The minimum size of a cluster, as a fraction of the total dataset size.

```
# Calculate cluster centroids
centroids = (
    data_with_clusters[data_with_clusters['Cluster_Label'] != -1]
    .groupby('Cluster_Label')
    .mean()
    .values
)
```

✓ 0.0s

```
# Calculate distance to cluster centroids
def calculate_distances(row, centroids):
    cluster_label = int(row['Cluster_Label'])
    if cluster_label == -1:  # Skip noise points
        return np.nan
    centroid = centroids[cluster_label]
    return euclidean(row[:-1], centroid)
```

✓ 0.0s

```
# Add distances to the DataFrame
data_with_clusters['Distance_To_Centroid'] = data_with_clusters.apply(
    lambda row: calculate_distances(row, centroids), axis=1
)
```

✓ 2.9s

```
# Display results
print(data_with_clusters)
```

✓ 0.0s

Figure 30. Computed centroid for each cluster and calculated with Euclidean distance

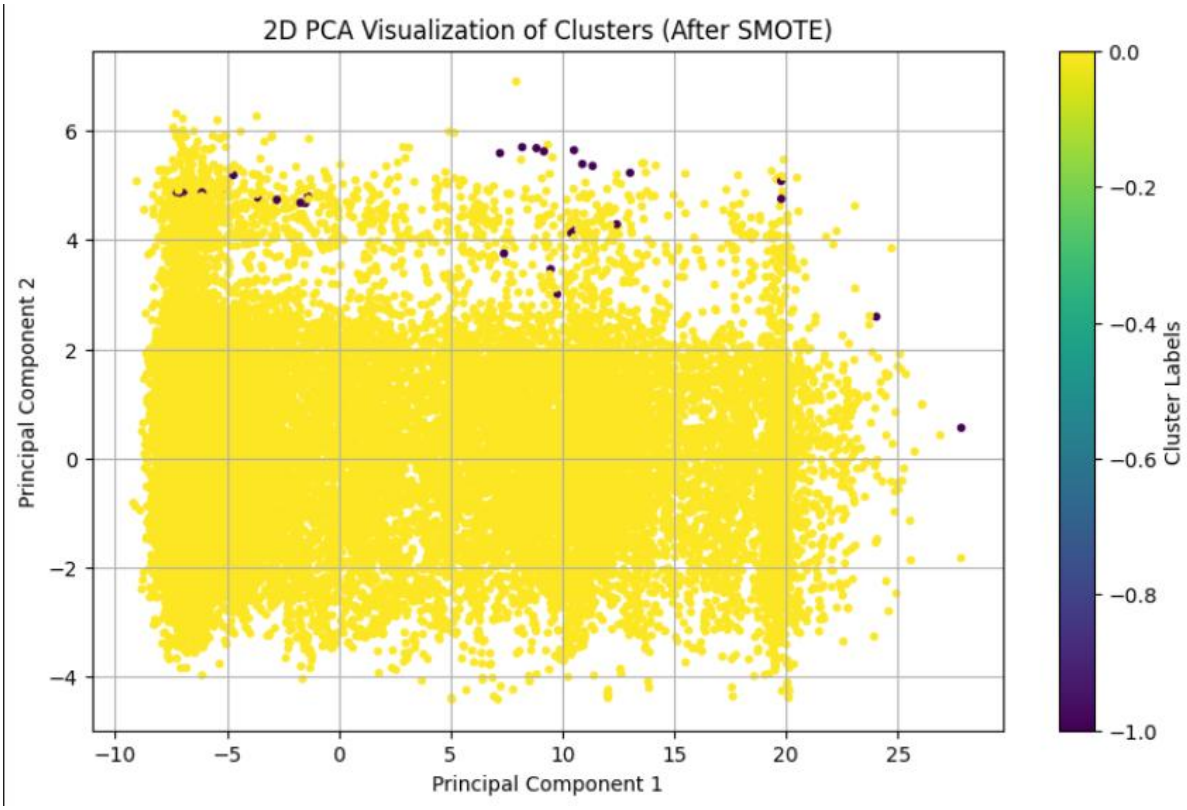

Figure 31. Visualization of clusters – 1 cluster

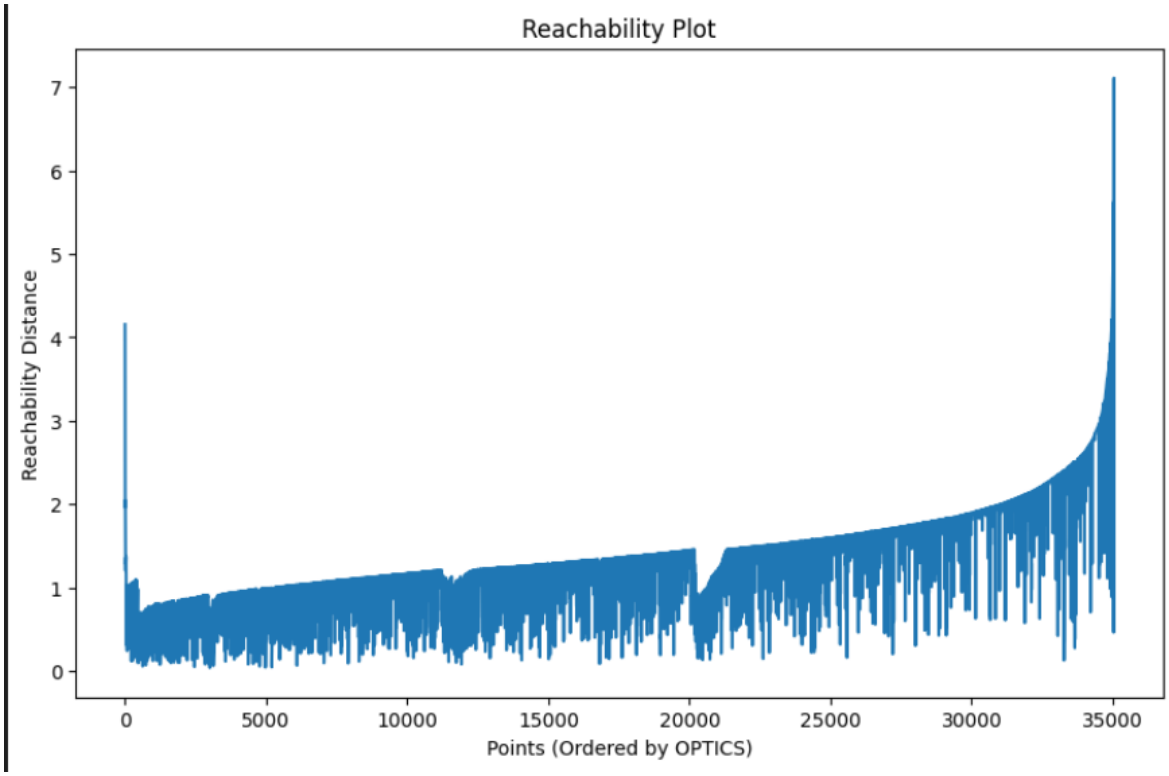

Figure 32. Reachability plot

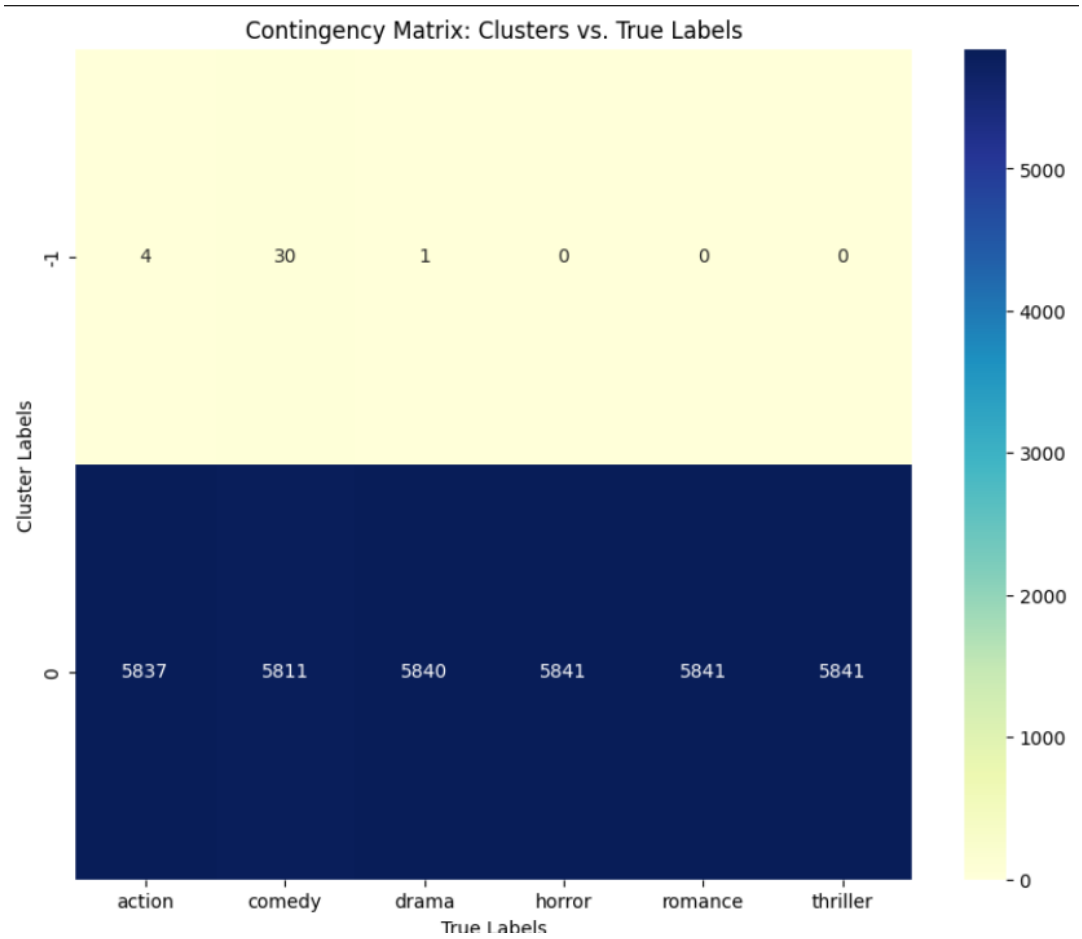

Figure 33. Contingency Plot

The rows represent cluster labels, and the columns represent true genre labels. Most points are clustered under one dominant cluster (Cluster 0) with very little differentiation across true genres. Cluster -1 represents noise points that do not belong to any cluster (OPTICS assigns -1 to outliers).

The clustering algorithm failed to identify meaningful clusters corresponding to true genres. All true labels are grouped into a single cluster (Cluster 0), indicating poor clustering performance.

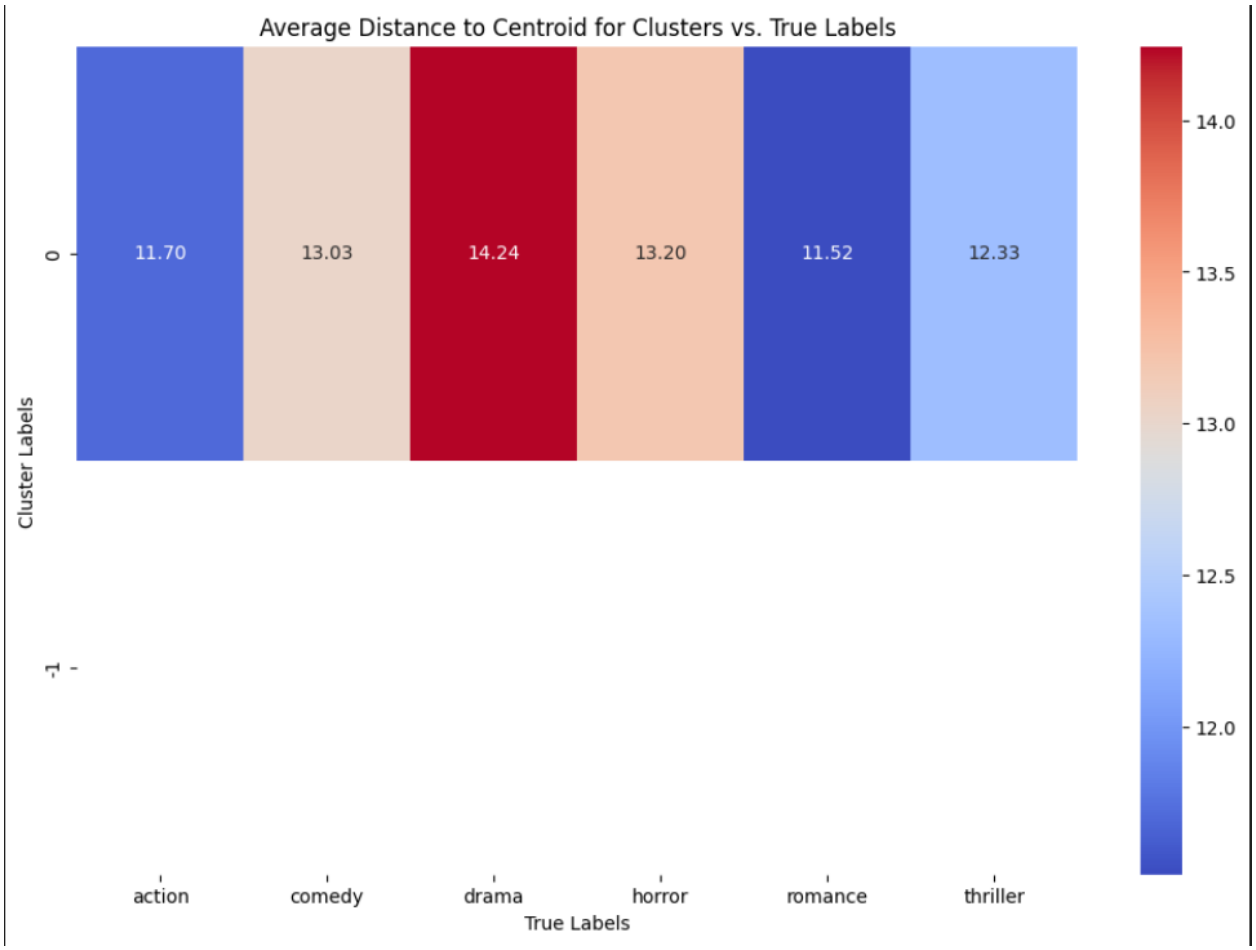

Figure 34. Average distance to centroid

Each bar represents the average distance from points in a given cluster to their cluster centroid for a specific true genre. Despite using a density-based algorithm, the distances for Cluster 0 vary across true genres ("drama" has a higher distance to centroid compared to "action").

Since all genres are predominantly grouped into a single cluster (Cluster 0), the variations in distances likely reflect how poorly the clustering captures genre-specific structure.

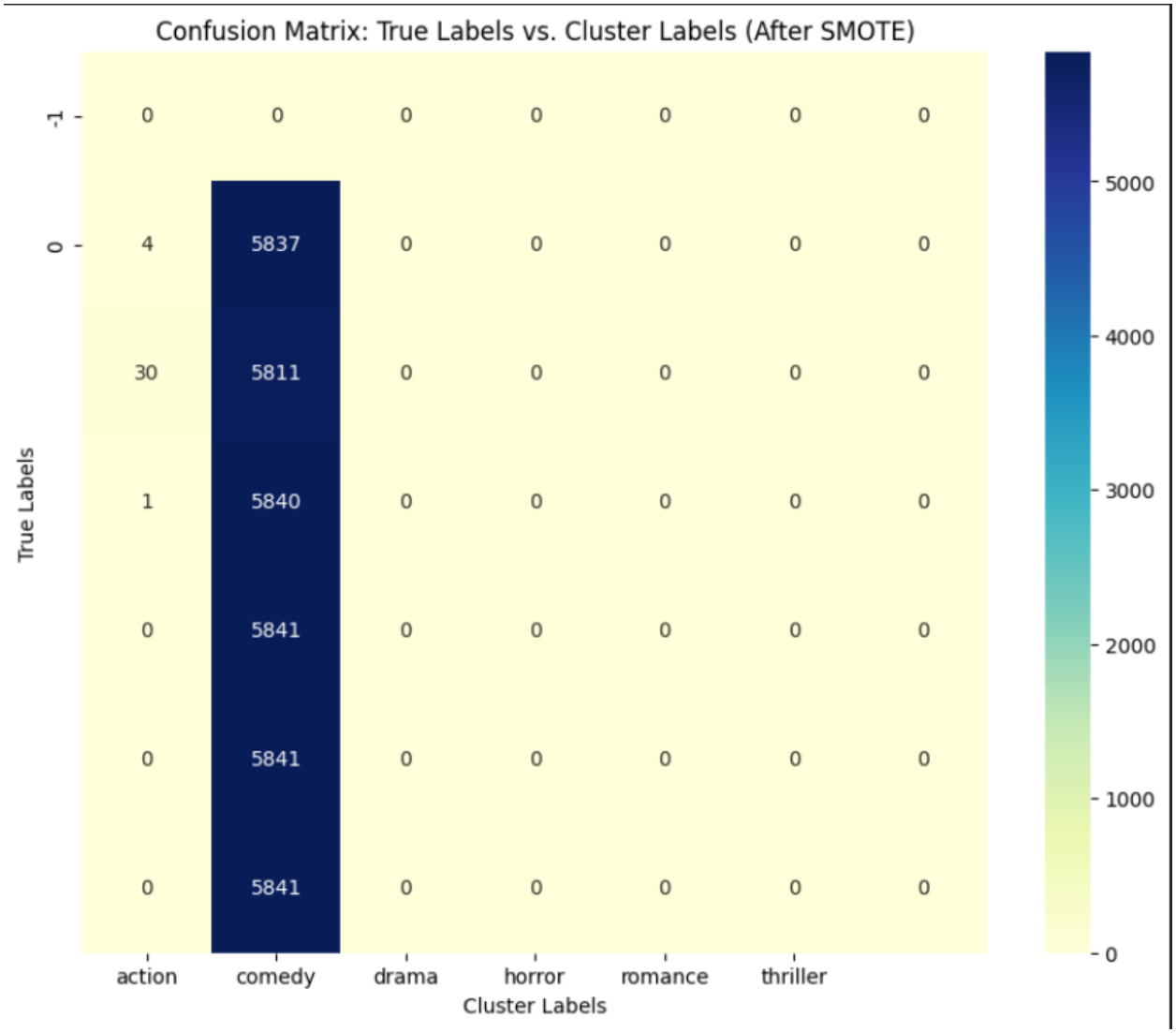

Figure 35. Confusion matrix

True labels are mapped to cluster labels. Like the first matrix, all true genres are predominantly mapped to a single cluster (Cluster 0). Noise points (cluster -1) are non-existent, as shown in the row for -1.

The clustering algorithm failed to differentiate between genres, as all genres map to a single cluster.

Adjusted Rand Index (ARI): 0.0000
Normalized Mutual Information (NMI): 0.0005

Second attempt:

```
# Apply OPTICS clustering
optics_model = OPTICS(min_samples=2, xi=0.1, min_cluster_size=0.08)
optics_model.fit(data_reduced)
```

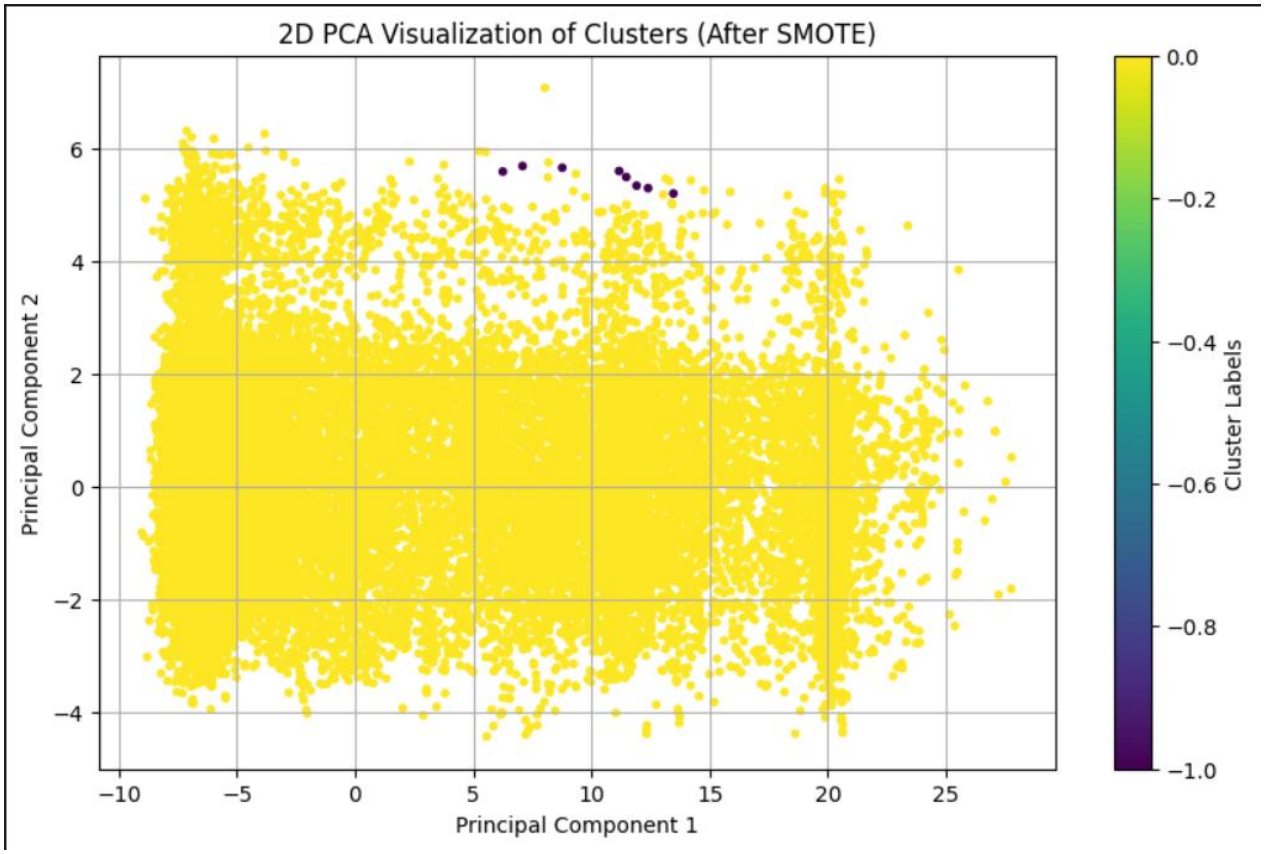

Figure 36. Visualization of clusters on second attempt

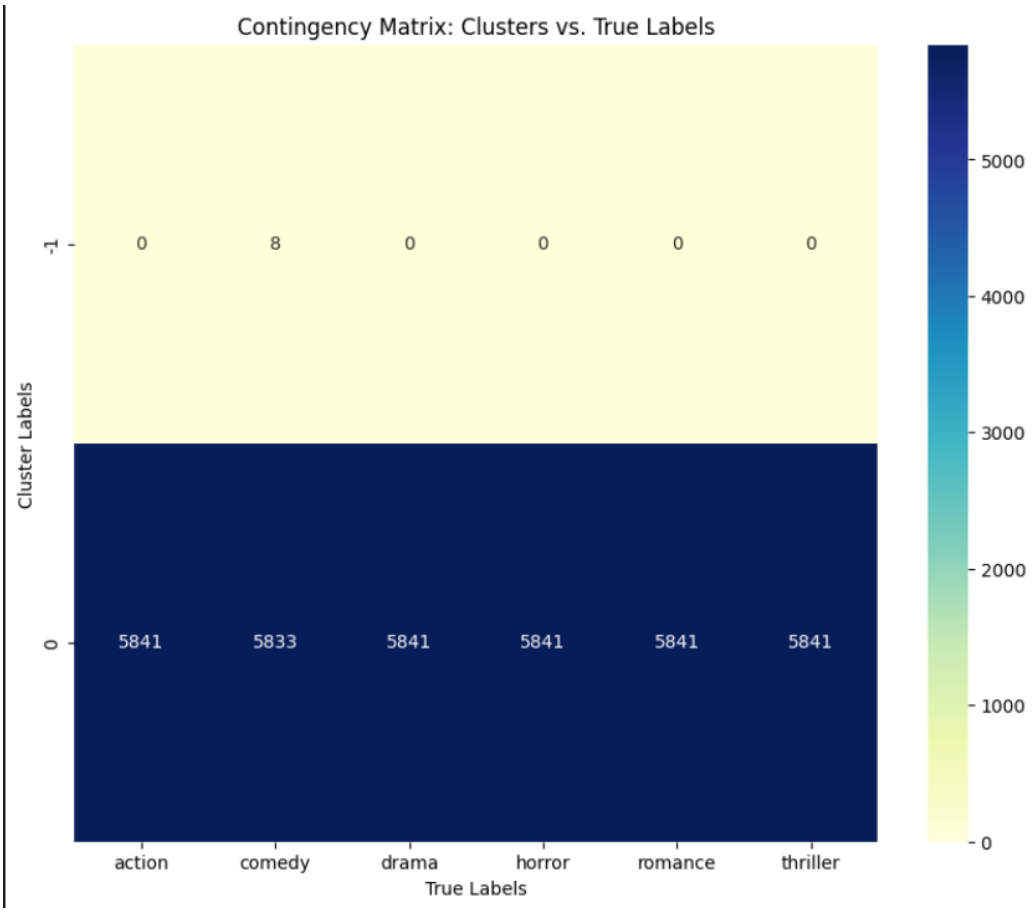

Figure 37. Contingency matrix on second attempt

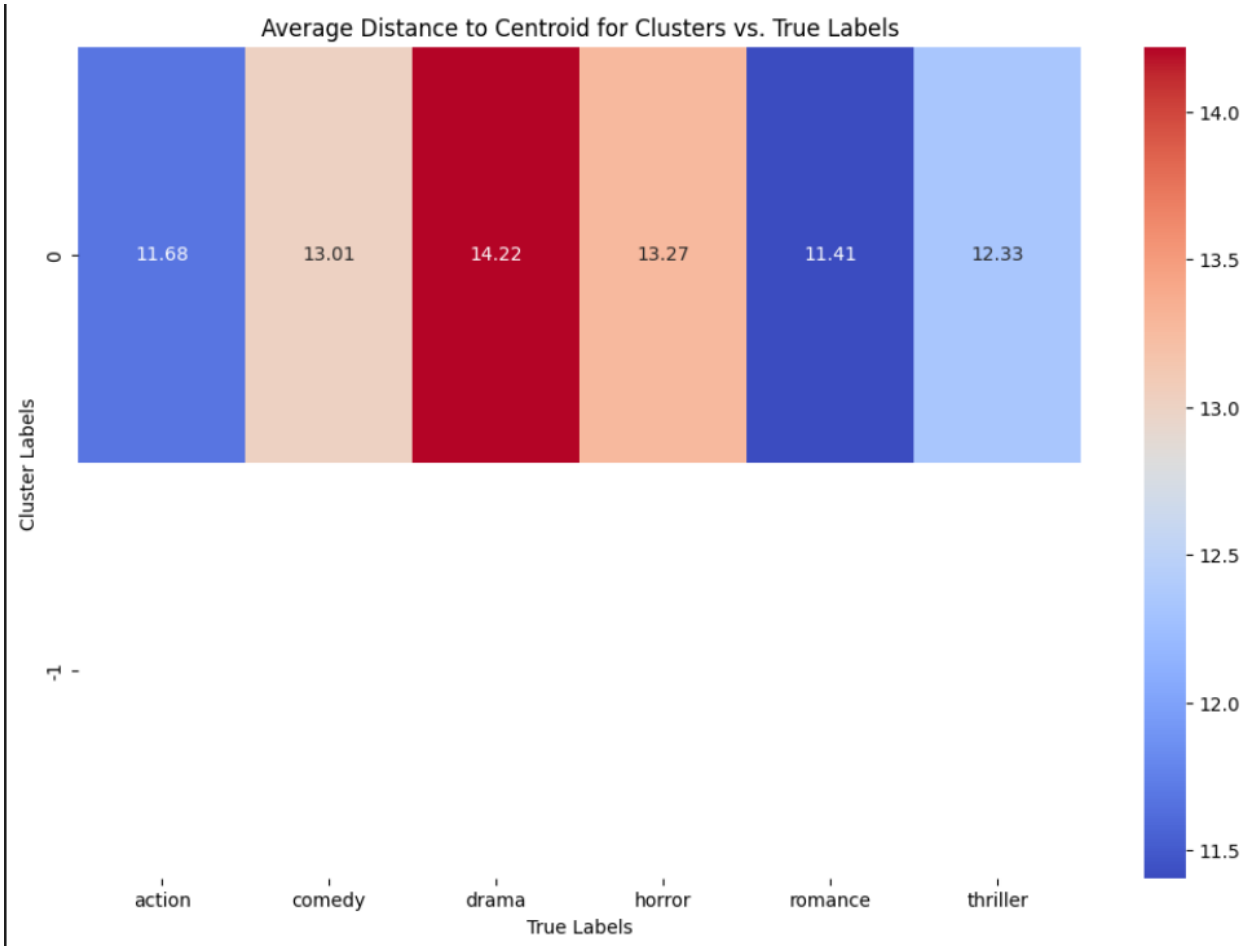

Figure 38. Average distance to centroid on second attempt

Supervised attempt: Random Forest Classification with Grid Search and Smote

The goal of this supervised method was to build a classification model (using *XGBoost*) to predict movie genres based on features derived from the dataset. It involved preprocessing the data, including handling categorical values, generating embeddings (Word2Vec) for textual features, balancing the training data using *SMOTE*, and optimizing the model with *GridSearchCV*.

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=42, k_neighbors=5)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)



# Standardize all features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)



# Define parameter grid for GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1],
    'n_estimators': [200, 300],
    'subsample': [0.8],
    'colsample_bytree': [0.8],
    'reg_alpha': [0.01, 0.1],
    'reg_lambda': [0.01, 0.1]
}



# Initialize the XGBoost classifier
xgb_model = xgb.XGBClassifier(objective='multi:softmax', num_class=len(np.unique(y_train)), seed=42)

# Perform grid search
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=3,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train_scaled, y_train_resampled)
```

Figure 39. Smote, XGBoost & GridSearch

```
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300, 'reg_alpha': 0.1, 'reg_lambda': 0.1, 'subsample': 0.8}
Best Cross-Validation Accuracy: 0.74
Validation Accuracy: 0.41
Test Accuracy: 0.40
```

Figure 40. Results with supervised attempt

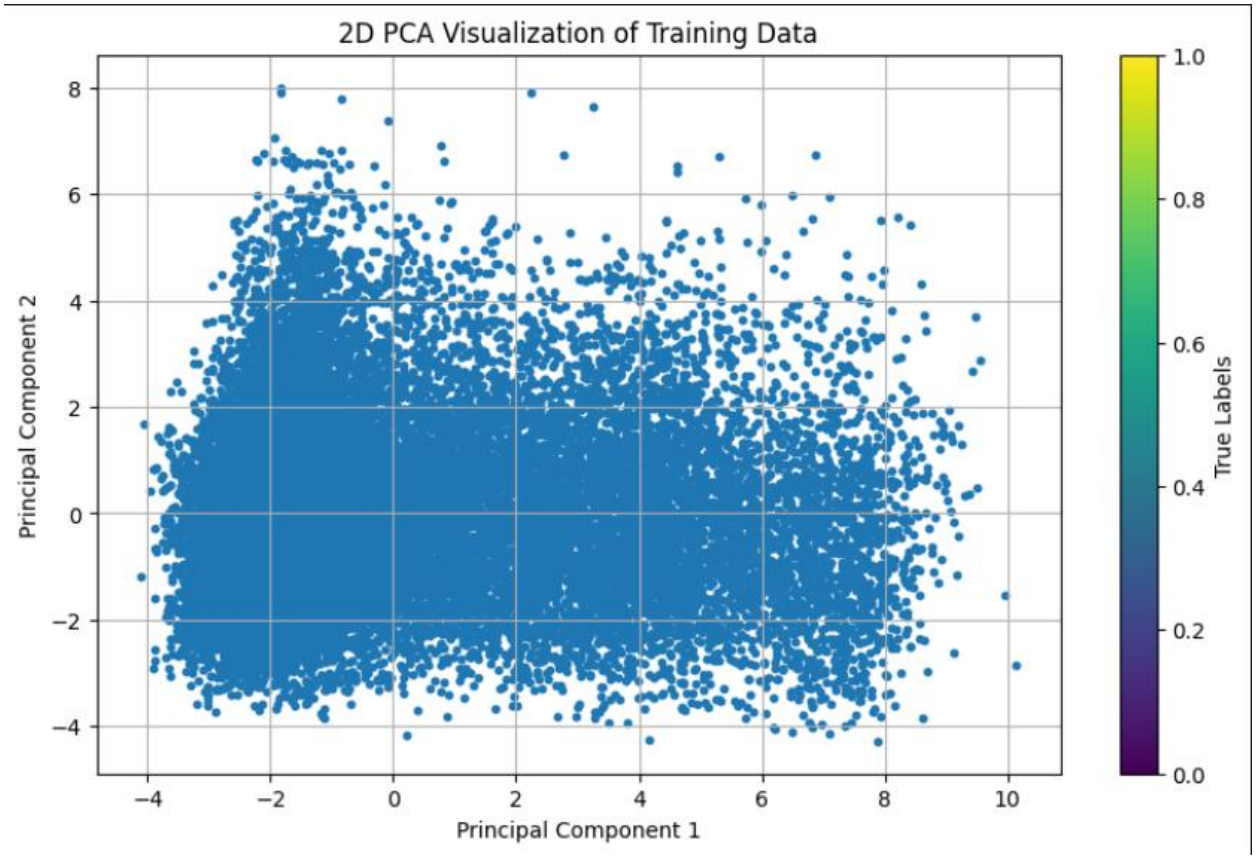

Figure 41. Visualization of training data on supervised attempt

Each point represents a sample from the training dataset. The points are densely packed in certain areas, suggesting overlap between different genres. There is no clear separation between classes, which may indicate that the features lack sufficient discriminatory power in 2D space.

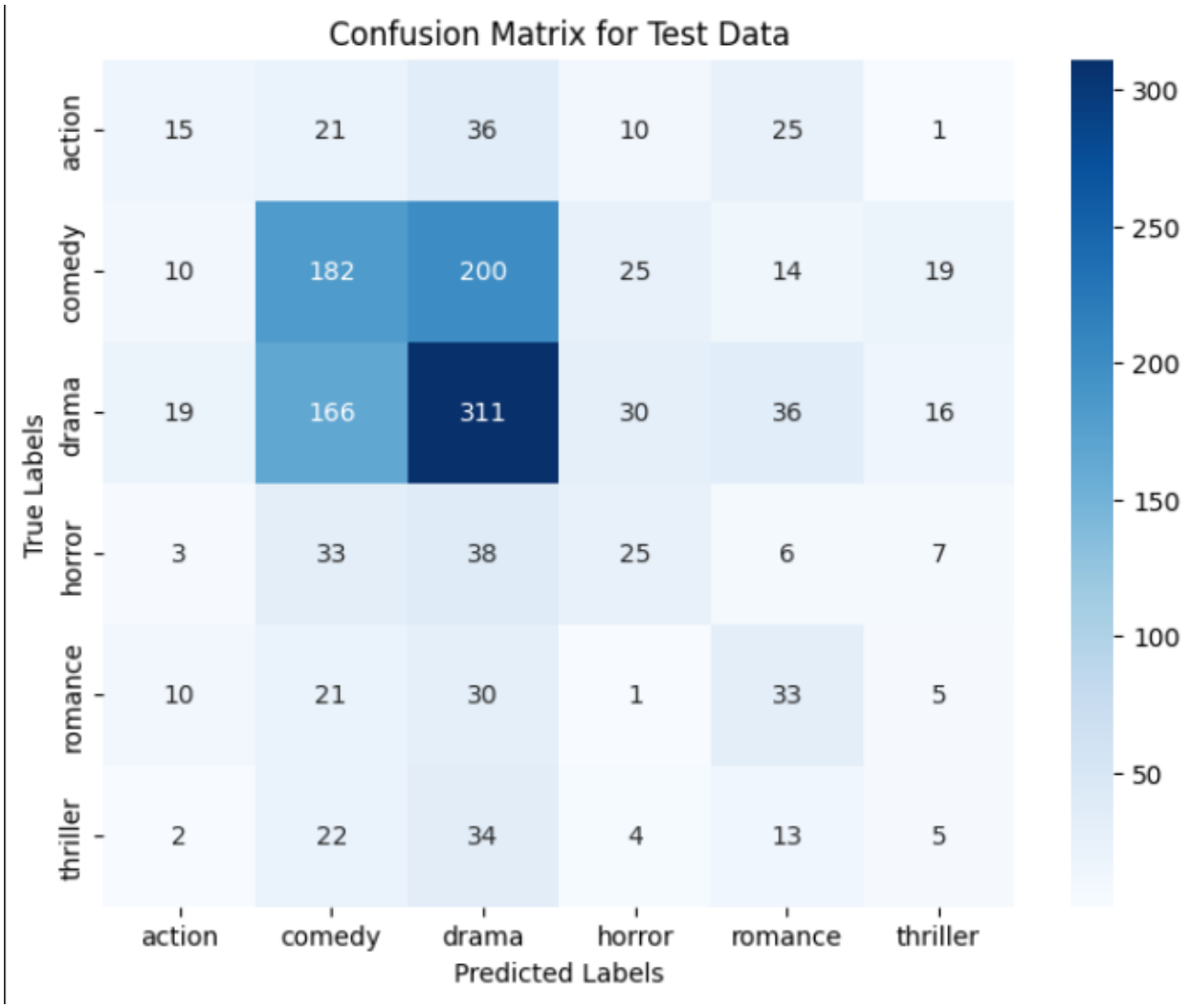

Figure 42. Confusion matrix on supervised attempt

Diagonal elements represent the correctly predicted sample for each genre. For example: Row 1, column 1 -182 samples were correctly classified; Row 2, column 2 – 311 samples were correctly classified.

Off-diagonal elements represent misclassifications. For example: Row 1, column 2 – 200 samples were incorrectly classified.

Random chance attempt: Cluster assignment and evaluation

The main goal was to establish a baseline performance of random clustering by evaluating random cluster-to-true label mapping. This serves as a reference to compare the performance of actual clustering algorithms, ensuring that meaningful clustering outperforms random assignment.

```
# Standardize all features
scaler = StandardScaler()
data_scaled = scaler.fit_transform(combined_features)

# Generate random cluster labels
true_labels = data['Genre']
random_cluster_labels = np.random.randint(0, len(np.unique(true_labels)), size=len(true_labels))
```

Figure 43. Random chance attempt

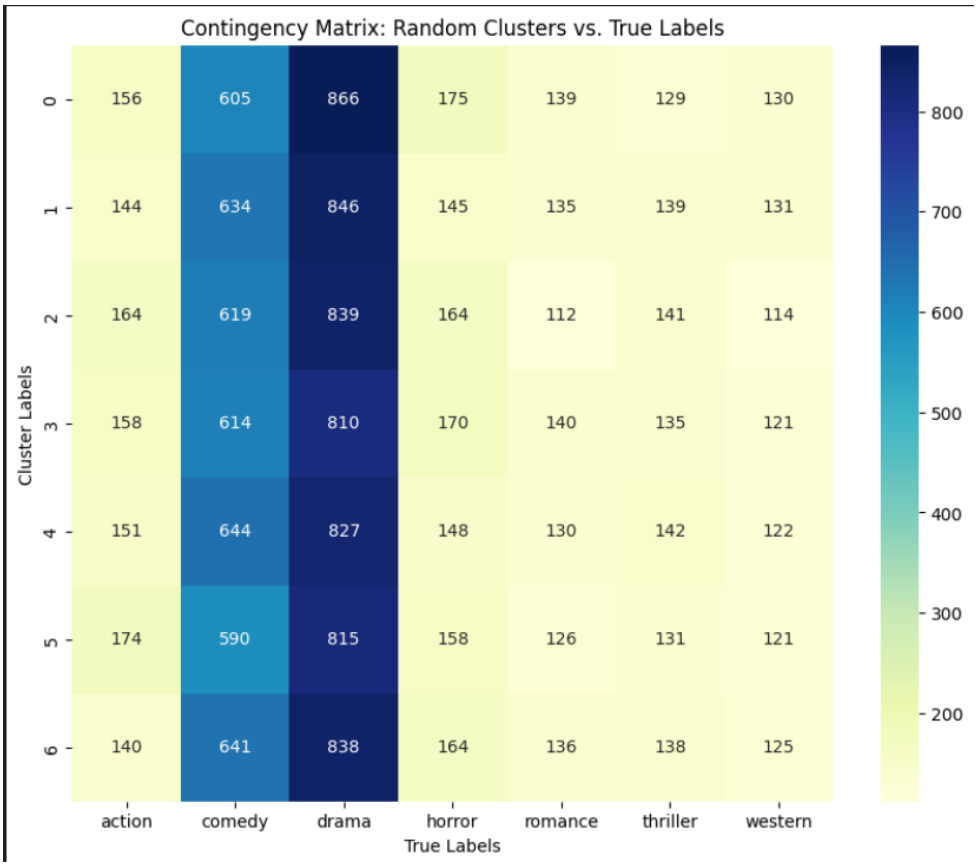

Figure 44. Contingency matrix on random chance attempt

Random Clustering ARI: -0.0001
Random Clustering NMI: 0.0004

Conclusions:

Random clustering performs poorly, as expected, providing a baseline for evaluation and metrics like ARI and NMI confirm no meaningful clustering, showing that random clustering has no predictive value and emphasizes the importance of achieving better alignment with true labels when using meaningful clustering algorithms.

Optics clustering fails to to separate genres, assigning nearly all samples to one cluster. Despite using SMOTE to balance the dataset, the clustering approach was ineffective in separating genres, possibly due to inadequate feature separability or clustering parameters.

Supervised model shows better alignment with true labels than clustering methods, but still struggles with overlapping genres(Comedy vs Drama). Indicates that supervised methods benefit from labeled data but require better feature representations to achieve higher accuracy.

Extra model – KMmeans clustering(on same features as in previous model)

It was used SMOTE to balance the dataset and StandardScaler for features.

```
n_clusters = len(np.unique(true_labels_balanced))
kmeans_model = KMeans(n_clusters=n_clusters, random_state=42, algorithm='lloyd', init='k-means++', n_init=20, max_iter=1000, tol=1e-4)
kmeans_labels = kmeans_model.fit_predict(data_reduced)
```

Figure 45. KMeans attempt

```
K-Means Results:
Silhouette Score: 0.2136
Adjusted Rand Index (ARI): 0.0290
Normalized Mutual Information (NMI): 0.0358
```

Figure 46. Results with KMeans

Silhouette score: 0.2136 indicates weak clustering structure, but is is better than 0. Points are not clearly separated and there is significant overlap between clusters.

Adjusted rand index: 0.0290 indicates very low alignment between the K-Means clusters and the true genre labels. This value is only slightly above random clustering (baseline), showing that K-Means struggled to find meaningful patterns in the data.

Normalizaed mutual information: 0.0358 is extremely low, indicating almost no shared information between the true labels and the clusters. This supports the conclusion that K-Means failed to identify meaningful genre groupings.

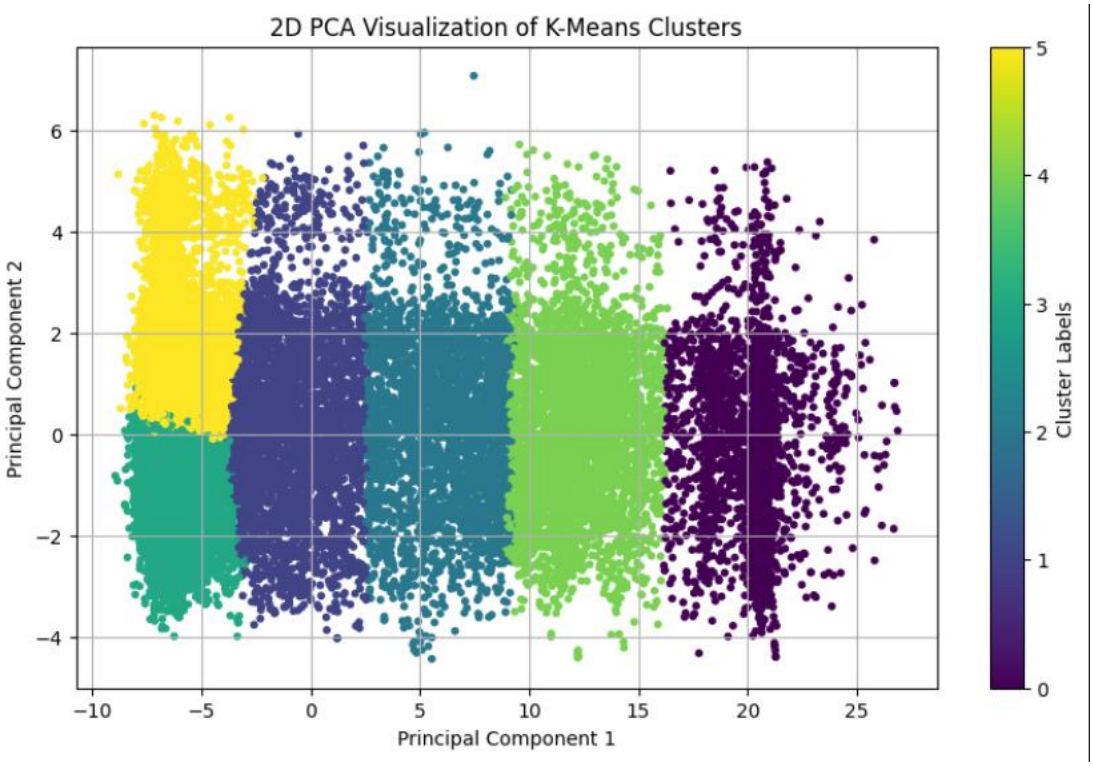

Figure 47. Visualization of clusters KMeans

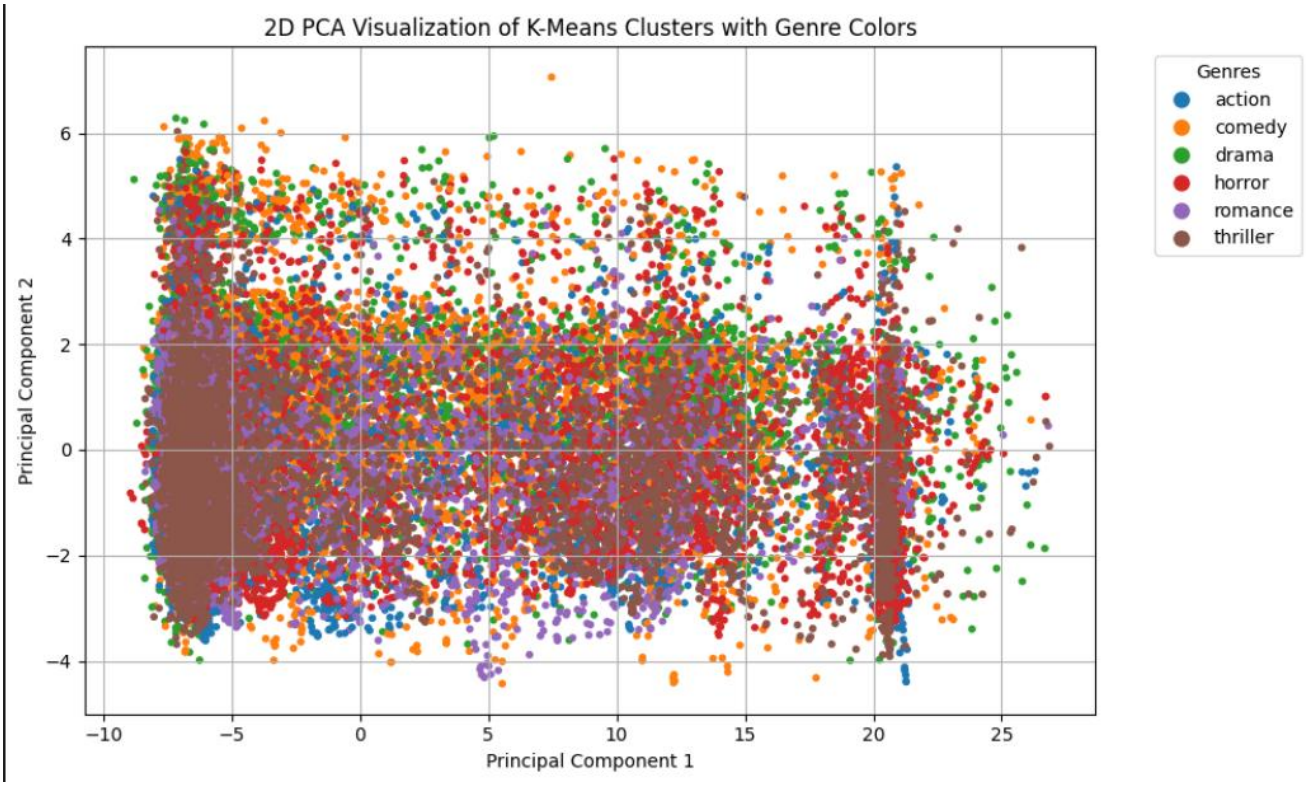

Figure 48. Visualization of clusters with Genre colors

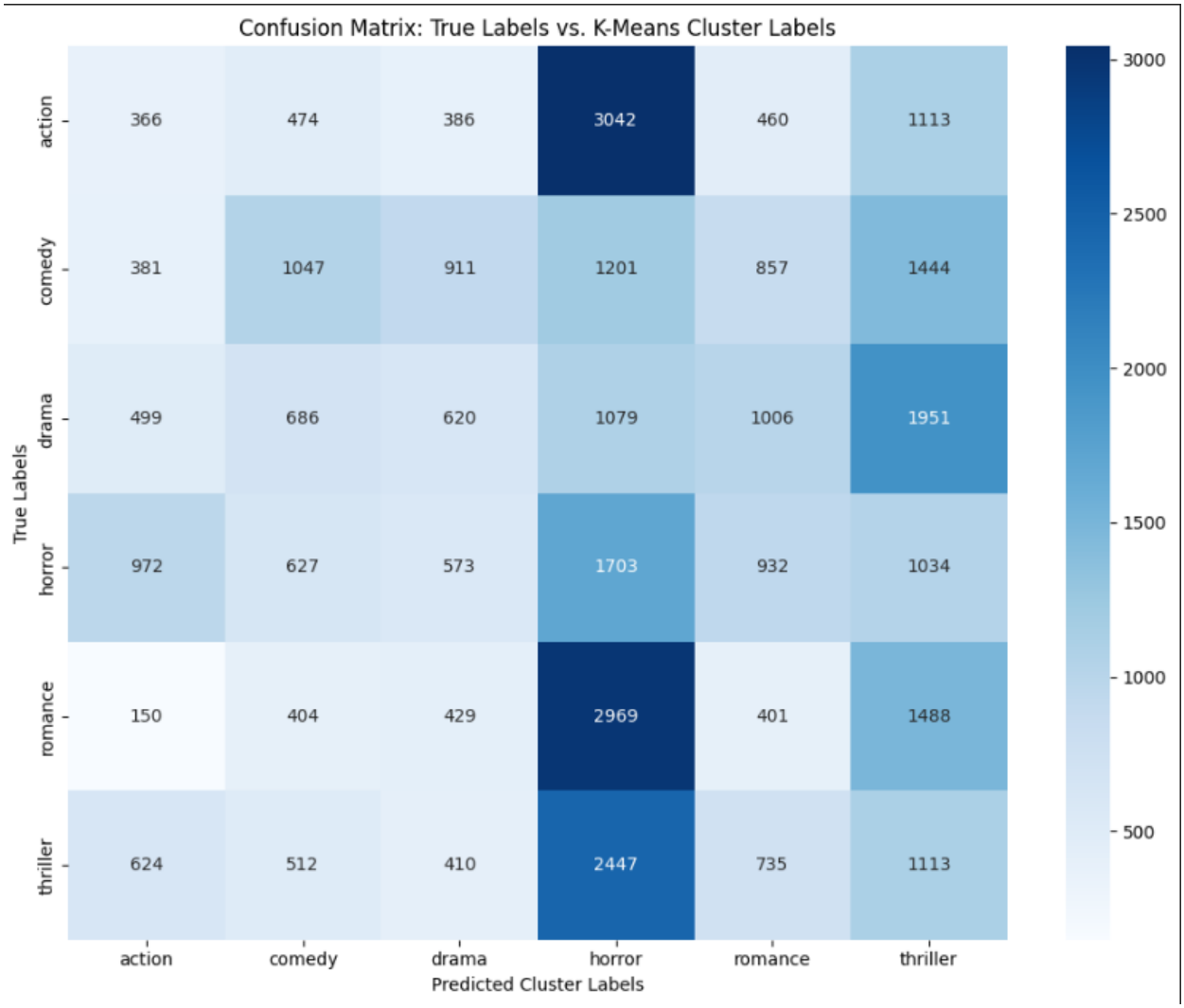

Figure 49. Confusion matrix KMeans model

Certain predicted clusters ("Horror") dominate across multiple genres. This indicates that the clustering algorithm is grouping many samples from different genres into the same cluster, leading to high overlap and less differentiation. Clusters do not represent distinct genres; instead, they capture overlapping characteristics from multiple genres, like "Drama," "Romance," and "Thriller" are dispersed across multiple clusters, indicating a lack of clear boundaries.