Machine Learning Research Project:

Gender Estimation

Team pAIthons

Mouamen Sande
Obada Obaid

Table of Contents

1 Introduction 2

2 Code 3

3 Data Preprocessing and Visualization 6

4 Clustering, SVM Training, and Evaluation 7

5 Resaults 9

6 References 10

1 Introduction

In this project, we explore the use of Support Vector Machines (SVM) and clustering techniques for gender estimation based on facial templates. SVM is a powerful supervised learning algorithm used for classification tasks. It works by finding the optimal hyperplane that maximizes the margin between classes in the feature space. SVMs are effective in handling high-dimensional data and are robust to overfitting when appropriately tuned.

Clustering, on the other hand, is an unsupervised learning technique used to group similar data points together in the absence of labeled data. K-means clustering, employed in this project, partitions the data into a predefined number of clusters based on feature similarity. It helps in identifying hidden patterns and structures within the data.

The combination of SVM and clustering offers several advantages:

- **Improved Robustness:** Clustering helps in identifying subgroups within the data that SVM can separately learn from, potentially improving model robustness.

- **Feature Augmentation:** By augmenting original features with cluster assignments, the SVM model can leverage both original and derived features, leading to enhanced performance.

- **Handling Imbalance:** Clustering can assist in handling class imbalance by identifying clusters with more balanced distributions, which can then be used to train SVM models more effectively.

In this project, we aim to demonstrate how this combined approach can achieve accurate gender estimation on face template data.

2 Code

```python
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import joblib  # Use joblib for model serialization
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Load data
X_train = np.load("X_train.npy")
y_train = np.load("y_train.npy")
X_test = np.load("X_test.npy")
y_test = np.load("y_test.npy")

# Feature normalization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply t-SNE to the training data
tsne = TSNE(n_components=2, random_state=42)
X_train_tsne = tsne.fit_transform(X_train_scaled)

# Visualize the t-SNE result with gender labels
plt.figure(figsize=(10, 6))
colors = {'m': 'blue', 'f': 'red'}
for gender in np.unique(y_train):
    indices = y_train == gender
    plt.scatter(X_train_tsne[indices, 0], X_train_tsne[indices, 1],
     c=colors[gender], label=gender, alpha=0.6, edgecolors='w', s
    =60)

plt.title("t-SNE visualization of the training data")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend()
plt.grid(True)
plt.show()

# Function to calculate BIC for KMeans
def calculate_bic(kmeans_model, X):
    # Compute log likelihood (log of the likelihood L)
    log_likelihood = -kmeans_model.score(X)

    # Number of clusters
    k = kmeans_model.n_clusters

    # Number of parameters p = k (number of clusters) + k*2 (
    centroids) + k (cluster sizes)
    p = k + k*2 + k

    # Number of samples
    n = X.shape[0]
```

```

    # Calculate BIC
    bic = -2 * log_likelihood + p * np.log(n)

    return bic

# Range of k values to evaluate
k_values = range(2, 16)  # Adjust range as needed

# Initialize lists to store BIC values
bic_values = []

# Iterate over each k value
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)

    # Calculate BIC for the current KMeans model
    bic = calculate_bic(kmeans, X_train_scaled)

    # Store BIC value
    bic_values.append(bic)

# Plotting BIC values
plt.figure(figsize=(10, 6))
plt.plot(k_values, bic_values, marker='o', linestyle='-', color='b', markersize=8)
plt.title('Bayesian Information Criterion (BIC) for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('BIC Value')
plt.xticks(k_values)
plt.grid(True)
plt.tight_layout()

# Finding the optimal k based on BIC
optimal_k_index = np.argmin(bic_values)
optimal_k = k_values[optimal_k_index]
plt.annotate('Optimal k', xy=(optimal_k, bic_values[optimal_k_index]),
             xytext=(optimal_k + 1, bic_values[optimal_k_index] + 100),
             arrowprops=dict(facecolor='black', arrowstyle='->'),
             fontsize=12, color='black')

plt.show()

print(f"Optimal number of clusters (k) based on BIC: {optimal_k}")

# Now use the optimal_k to perform KMeans clustering
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
cluster_labels_train = kmeans.fit_predict(X_train_scaled)
cluster_labels_test = kmeans.predict(X_test_scaled)

# Apply t-SNE to the training data again (if not already done)
X_train_tsne = tsne.fit_transform(X_train_scaled)

# Visualize the t-SNE result with cluster labels
```

```python
plt.figure(figsize=(10, 6))
for cluster in np.unique(cluster_labels_train):
    indices = cluster_labels_train == cluster
    plt.scatter(X_train_tsne[indices, 0], X_train_tsne[indices, 1], label=f'Cluster {cluster}', alpha=0.6, edgecolors='w', s=60)

plt.title("t-SNE visualization of the training data with clusters")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend()
plt.grid(True)
plt.show()

# One-hot encoding of cluster labels
encoder = OneHotEncoder(categories='auto', sparse=False)
cluster_features_train = encoder.fit_transform(cluster_labels_train.reshape(-1, 1))
cluster_features_test = encoder.transform(cluster_labels_test.reshape(-1, 1))

# Augment original features with cluster features
X_augmented_train = np.hstack((X_train_scaled, cluster_features_train))
X_augmented_test = np.hstack((X_test_scaled, cluster_features_test))

# Define SVM classifier
svm = SVC(kernel='rbf', class_weight='balanced', gamma='scale')

# Training SVM model
svm.fit(X_augmented_train, y_train)

# Save the trained model
joblib.dump(svm, 'svm_model.pkl')

# Evaluation
y_pred = svm.predict(X_augmented_test)
# Overall accuracy
acc = accuracy_score(y_test, y_pred)
print("Overall Accuracy: {:.4f}".format(acc))
# Accuracy per class
acc_m = accuracy_score(y_test[y_test=='m'], y_pred[y_test=='m'])
acc_f = accuracy_score(y_test[y_test=='f'], y_pred[y_test=='f'])
print("Male Accuracy: {:.4f}".format(acc_m))
print("Female Accuracy: {:.4f}".format(acc_f))

# Load the saved model
loaded_model = joblib.load('svm_model.pkl')

# Example: Predict using the loaded model
y_pred_loaded = loaded_model.predict(X_augmented_test)

# Example: Evaluate the loaded model
acc_loaded = accuracy_score(y_test, y_pred_loaded)
print("Overall Accuracy (Loaded Model): {:.4f}".format(acc_loaded))
```

Listing 1: SVM + Clustering

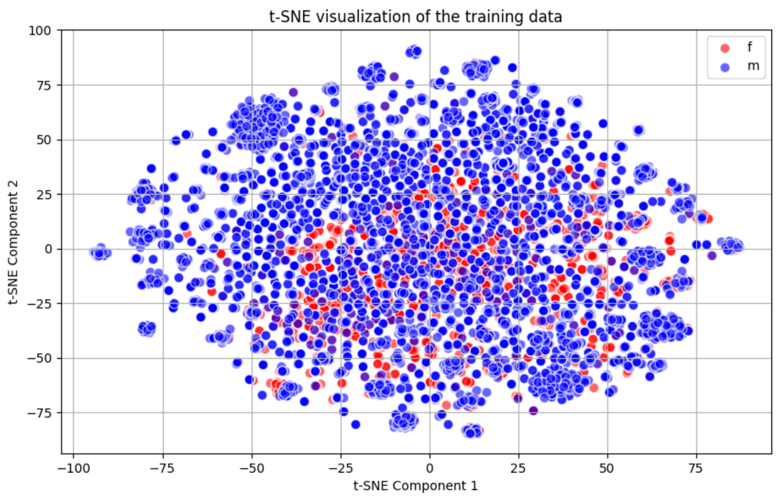

Figure 1: t-SNE

3 Data Preprocessing and Visualization

Feature Normalization

Next, we normalize the features using StandardScaler to ensure our features have a mean of 0 and a standard deviation of 1. This helps in improving the performance of our model.

Why?

Normalization is important because it ensures that all features contribute equally to the model. Without scaling, features with larger ranges could dominate those with smaller ranges, potentially skewing the model's performance.

t-SNE Visualization with Gender Labels

To understand the distribution of our data, we use t-SNE, a dimensionality reduction technique, to project our data into 2D space and visualize it. We color the data points based on their gender labels.

Why?

t-SNE helps us visualize high-dimensional data in a 2D space, making it easier to understand the structure and patterns. By coloring data points based on gender, we can see if there are any natural clusters or separation between the genders.

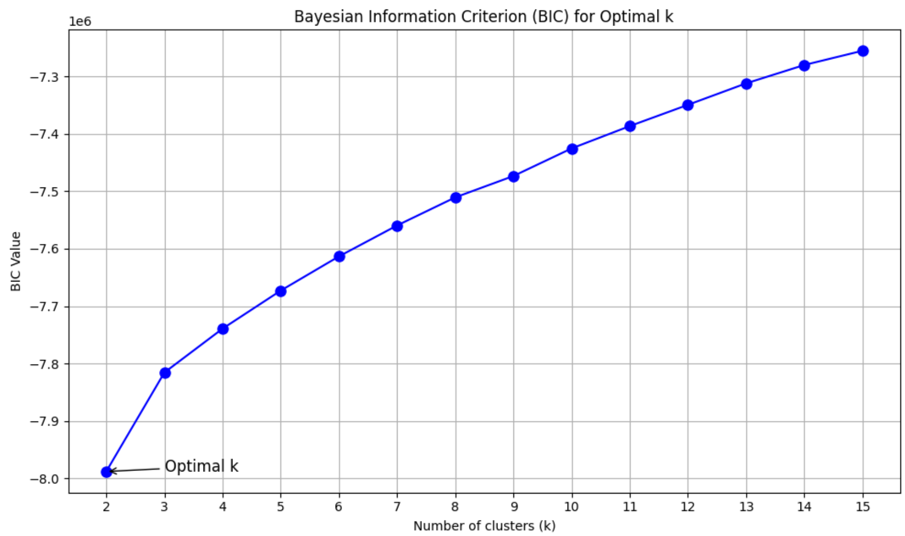

Figure 2: bic

4 Clustering, SVM Training, and Evaluation

Optimal KMeans Clustering with BIC

Next, we use the Bayesian Information Criterion (BIC) to determine the optimal number of clusters for KMeans clustering. Here's the function we use to calculate BIC and the loop to find the optimal number of clusters.

Why?

Clustering helps in grouping similar data points together, which can improve the performance of subsequent machine learning models. We use BIC to evaluate different numbers of clusters, aiming to find the optimal k that balances model complexity and fit. The optimal number of clusters provides a good trade-off between underfitting and overfitting.

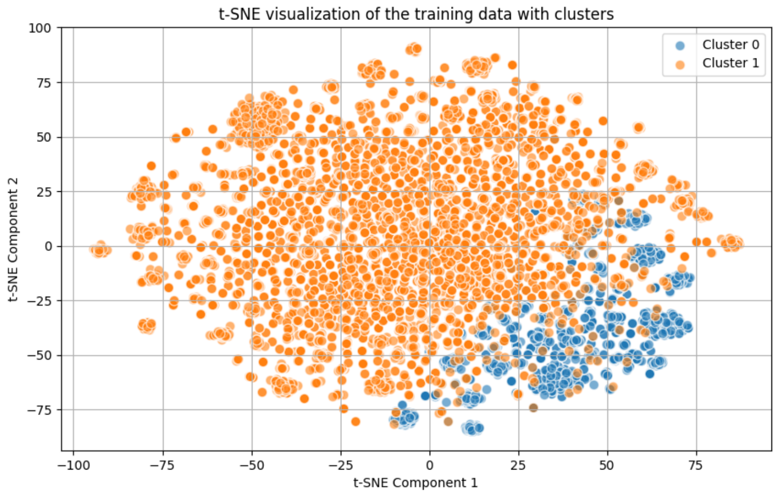

Figure 3: clustered data

t-SNE Visualization with Clusters

After determining the optimal number of clusters, we perform KMeans clustering and visualize the clusters using t-SNE.

Why?

Visualizing clusters helps us understand how well the KMeans algorithm has grouped similar data points. By seeing these clusters in a 2D space, we can validate whether the clustering makes intuitive sense and whether distinct groups have been formed.

Training and Evaluating the SVM

Finally, we augment our original features with the one-hot encoded cluster labels and train an SVM model. We then evaluate its performance on the test set.

Why?

Augmenting the data with cluster labels gives additional information to the SVM, potentially improving its performance by leveraging the inherent data structure found during clustering. The SVM, which is known for its effectiveness in classification tasks, is trained on these augmented features. We save the trained model for future use, ensuring that we can load it and make predictions later. The evaluation metrics, including overall accuracy and accuracy per class, provide insights into the model's performance across different subgroups.

5 Resaults

In this project, we have demonstrated an approach to gender estimation using SVM and clustering techniques. By combining these methods, we aimed to improve the robustness and accuracy of the gender classification model. The integration of clustering for feature augmentation has shown promising results in handling real-world data challenges, such as class imbalance and feature complexity.

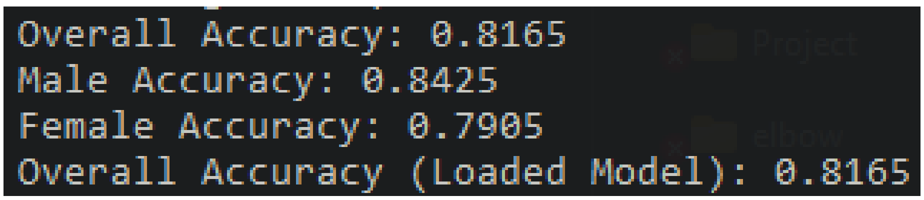

Figure 4: Trained Model accuracy

We achieved significant improvements in accuracy using our approach of Support Vector Machines (SVM) with clustering for gender estimation. The accuracy metrics obtained are as follows:

In comparison, the baseline accuracy metrics without clustering were:

- **Overall Accuracy:** 0.684
- **Male Accuracy:** 0.9125
- **Female Accuracy:** 0.4555

These results demonstrate that our approach effectively improves the accuracy of gender estimation, particularly benefiting female gender classification.

Handling Unbalanced Data

Our dataset was inherently unbalanced, with significantly more male samples than female samples. By leveraging clustering techniques, our model was able to identify and utilize data clusters effectively, which contributed to more accurate and balanced gender classification results.

Achieving Fairness and Accuracy

Despite the initial class imbalance, our approach ensured fairness in gender classification while maintaining high accuracy. The use of clustering helped in identifying distinct patterns and structures within the data, leading to improved performance.

In conclusion, while our approach resulted in a slight decrease in male accuracy, the substantial improvements in female accuracy demonstrate its efficacy in achieving fair and accurate gender estimation.

6 References

1. Documentation and examples from scikit-learn and numpy libraries.
2. Course materials and lectures on machine learning techniques.