DATA604 – Midterm 2

Introduction

In this project, I use the Fashion MNIST dataset. The Fashion MNIST dataset consists of 60,000 training images and 10,000 test images across 10 different clothing categories. Each image is a grayscale image of fashion items of size 28x28 pixels, which is flattened into a 784-dimensional feature vector. The dataset was taken from Zalando Research's GitHub repository.

Link: https://github.com/zalandoresearch.

The 10 class labels in the dataset are as follows (Class — Category):

0 — T-shirt/top, 1 — Trouser, 2 — Pullover, 3 — Dress, 4 — Coat, 5 — Sandal, 6 — Shirt, 7 — Sneaker, 8 — Bag, 9 — Ankle boot

This report explores the application of Principal Component Analysis (PCA), k-Nearest Neighbors (kNN) classification, and Support Vector Machines (SVM) on the Fashion MNIST dataset and to identify which model is a better fit on the dataset.

The study investigates how varying the number of neighbors (k) affects both global and local accuracy in kNN, and how different kernels impact performance in SVM. Global accuracy measures the overall correctness of the model across all classes, calculated as the total number of correctly classified samples in the test dataset divided by the total test samples. Local accuracy refers to the accuracy within a specific class, indicating how well the model classifies instances of a particular class.

Principal Component Analysis (PCA)- Working

PCA is a dimensionality reduction technique that transforms the original features into a new set of orthogonal components ranked by the amount of variance they capture from the data. It helps reduce the number of features while preserving as much information as possible.

k-Nearest Neighbors (kNN) – Working

kNN is a simple, instance-based learning algorithm that classifies a sample based on the majority label among its k nearest neighbors in the training set.

Support Vector Machines (SVM) – Working

SVM is a supervised learning algorithm that finds the hyperplane that best separates data points of different classes with the maximum margin. It can handle non-linear classification using kernel functions that projects data into higher-dimensional spaces.

Methodology

The Fashion MNIST dataset was first standardized across all features as classification task is being performed. PCA was applied to the dataset with n_components = 784 (the same as the number of features). After applying PCA, top components were selected to reduce the dimensionality of the dataset, thereby decreasing the computational complexity. The kNN classifier performance was observed on the PCA-reduced training set for different values of k. Per-class success rates were also computed for each value of k. Subsequently, SVM classifier performance was observed on the PCA-reduced training set for different kernels and per-class success rates were computed.

Question 1: Perform PCA

Sub-question A: From a mathematical stand point what does $T : R^{784} \rightarrow R^{784}$, $Tx = Px$ do?

From a mathematical standpoint, the transformation $T: R^{784} \rightarrow R^{784}$, defined as $T(x)=Px$, represents a linear change of basis in the 784-dimensional feature space. P is the orthogonal matrix whose columns are the principal components obtained from the covariance matrix of the standardized dataset. The matrix P is of size 784x784. Although the dimensionality in the transformation is kept the same, the data is now represented in a new coordinate system where the axes align with

directions of maximum variance. This transformation preserves all the information from the original dataset.

Sub-question B: Display the original image and the transformed image. Address whether the new images resemble?

Randomly two images were selected from each class. The original and transformed images were displayed. The transformed images were reconstructed using the principal component matrix. Upon comparing the original and transformed images, it is observed that the transformed images largely preserved the shape of the original images. As all the number components is equal to the number of features, few minor distortions are noticeable due to the change of basis, but the overall resemblance is clear, and the class identity of the images remains recognizable. This observation confirms that PCA transformation retains the key information about the original samples even though it re-expresses them in a different coordinate system.

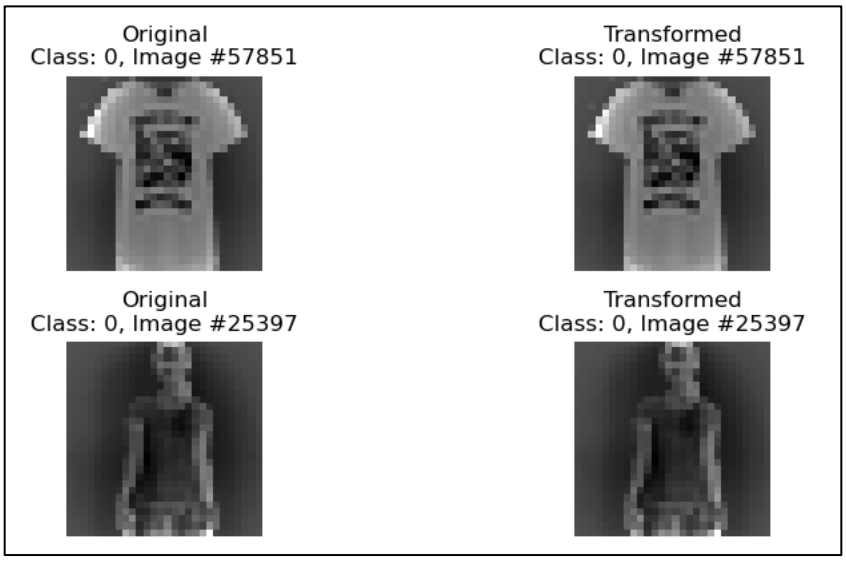


Figure 1: Original and reconstructed images of Class 0 after performing PCA

Sub-question C: Pick optimal number of principal components between [1,50]?

A graph of cumulative explained variance against the number of components in the range [1,50] was plotted to determine the optimal number of components. The curve starts to flatten after n_components = 10. This indicates that additional components do not contribute much to capturing the variance. Even at 50 principal components, the cumulative explained variance is below the desired threshold of 90%. It is preferred to keep more than 90% of the data, but there is a constraint to choose the number of components. Therefore, I choose 50 principal components as the optimal number for dimensionality reduction. This choice reduces dimensionality by more than 93% (from 784 to 50). Choosing more components would increase computational cost without substantial gains in explained variance. The Cumulative Explained Variance is 0.8007 (80.07%) at n_components = 50.

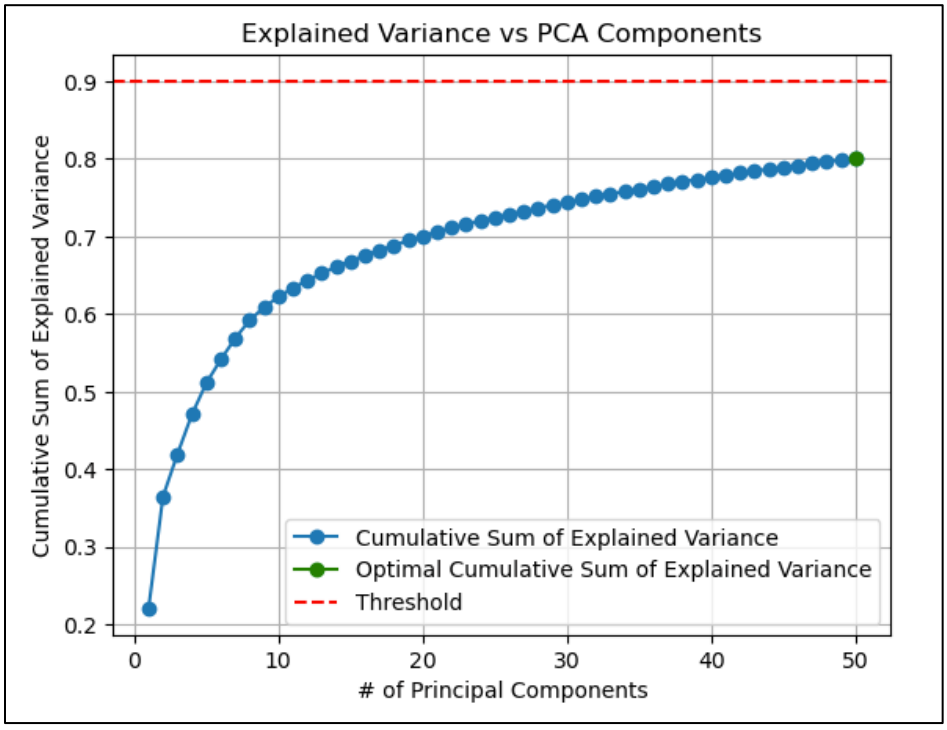


Figure 2: Graph of Cumulative Explained Variance vs Number of Principal Components

Question 2: Perform kNN

After applying PCA and selecting the top 50 principal components, the Fashion MNIST dataset was used to train and evaluate a k-Nearest Neighbors (kNN) classifier. The training set consisted of the first 6000 images from each class (total 60,000 images), and the testing set consisted of the 1000 images from each class (total 10,000 images). I evaluated the model for different values of k in {5,10,15,20}, using the Euclidean distance metric.

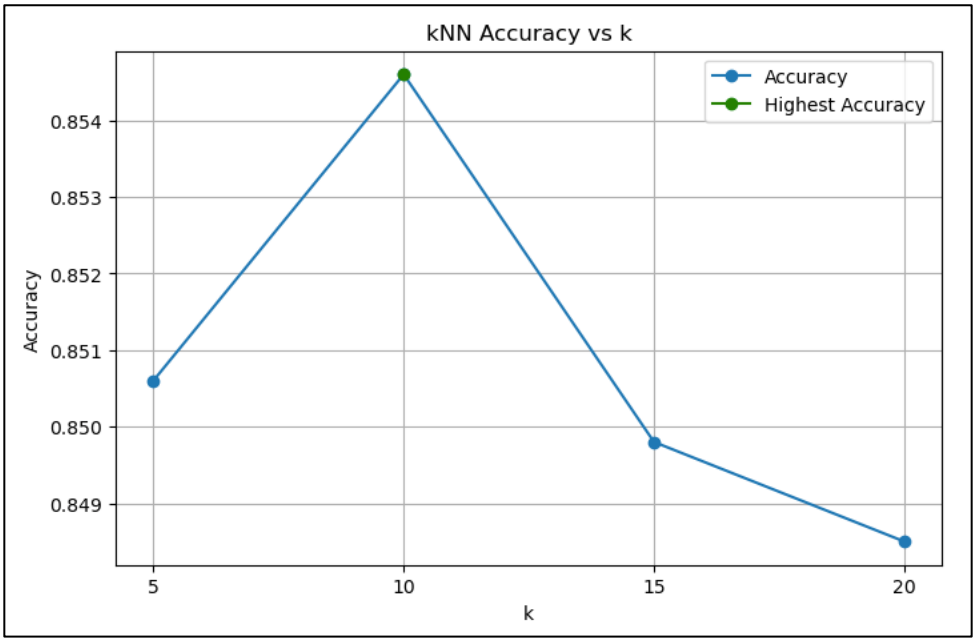


Figure 3: Graph of kNN model accuracy vs values of k

k	Accuracy %
5	85.06%
10	85.46%
15	84.98%
20	84.85%

Table 1: kNN model accuracy vs values of k

```
Results for k=10
Overall Accuracy: 0.8546
Success Rates per Class:
              precision    recall  f1-score   support

           0     0.7812    0.8460    0.8123      1000
           1     0.9876    0.9570    0.9721      1000
           2     0.7727    0.7920    0.7822      1000
           3     0.8695    0.8660    0.8677      1000
           4     0.7543    0.7830    0.7684      1000
           5     0.9632    0.8900    0.9252      1000
           6     0.6480    0.5800    0.6121      1000
           7     0.8913    0.9350    0.9126      1000
           8     0.9653    0.9470    0.9561      1000
           9     0.9135    0.9500    0.9314      1000

    accuracy                         0.8546     10000
   macro avg     0.8547    0.8546    0.8540     10000
weighted avg     0.8547    0.8546    0.8540     10000
```

Figure 4: Success Rate of each class at k=10

The highest overall accuracy of 85.46% was achieved at k=10.

Comparing the kNN classification results before and after applying PCA, the following observations are made:

- In Midterm 1, the highest overall global accuracy achieved was 86.17% at k=4 without using PCA. In contrast, after applying PCA in Midterm 2, the highest global accuracy achieved was 85.46% at k=10.
- In terms of local (per-class) accuracy, the best performing class in both cases was Class 1 (Trouser). Without PCA, Class 1 achieved a local accuracy of 96.9%, while after PCA, it slightly decreased to 95.7%.
- Interestingly, the worst performing class, Class 6 (Shirt), showed a slight improvement: the local accuracy increased from 55.2% without PCA to 58.0% after applying PCA.
- Overall, applying PCA resulted in a small decrease in global accuracy, but it helped to slightly improve the classification performance of the most difficult class, while significantly reducing computational complexity.
- This is expected since some data is inevitably lost during dimensionality reduction.

Question 3: Perform SVM

After applying PCA and selecting the top 50 principal components, the Fashion MNIST dataset was used to train and evaluate a k-Nearest Neighbors (kNN) classifier. The classification was performed in a multi-class setting using the one-vs-rest strategy, which is the default approach in scikit-learn's SVM implementations. Both a linear kernel and an RBF (Radial Basis Function) kernel are tested to evaluate the impact of different kernel choices on model performance.

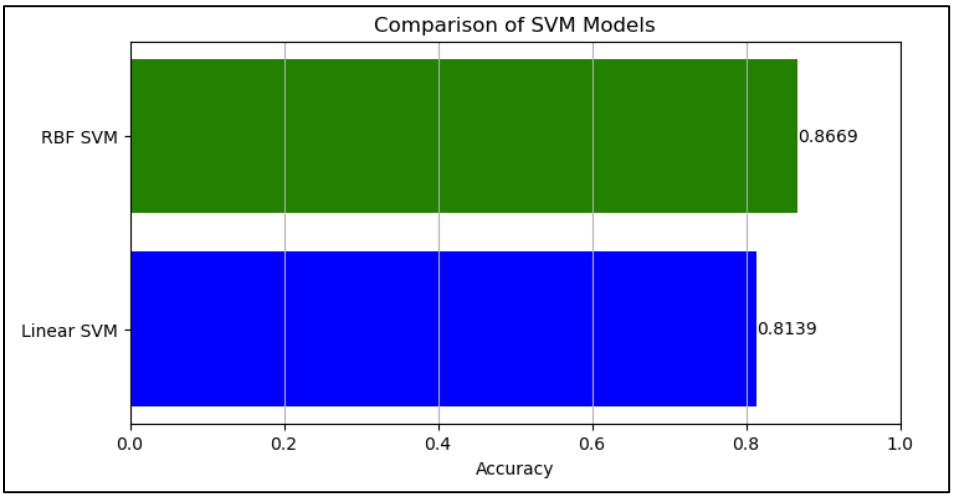


Figure 5: Comparison of SVM Models

The best performance was achieved using the RBF kernel with global accuracy of 86.69%.

Results for SVM using RBF kernel:
Overall Accuracy: 0.8669
Success Rates per Class:

	precision	recall	f1-score	support
0	0.8144	0.8380	0.8260	1000
1	0.9917	0.9560	0.9735	1000
2	0.7838	0.7720	0.7778	1000
3	0.8523	0.8890	0.8703	1000
4	0.7816	0.7980	0.7897	1000
5	0.9616	0.9260	0.9435	1000
6	0.6788	0.6360	0.6567	1000
7	0.9082	0.9400	0.9238	1000
8	0.9605	0.9730	0.9667	1000
9	0.9317	0.9410	0.9363	1000
accuracy			0.8669	10000
macro avg	0.8665	0.8669	0.8664	10000
weighted avg	0.8665	0.8669	0.8664	10000

Figure 6: Success Rate of each class in SVM with RBF Kernal

The RBF (Radial Basis Function) kernel outperformed the linear kernel as it is capable of capturing non-linear relationships in the data. While PCA reduced the dimensionality of the dataset, the underlying class boundaries in Fashion MNIST remained non-linearly separable in many cases. The RBF kernel's ability to project data into a higher-dimensional space allows it to form more flexible decision boundaries, leading to improved classification performance compared to the linear kernel.

Summary

In this project, I applied Principal Component Analysis (PCA) to the Fashion MNIST dataset to reduce its dimensionality from 784 features to 50 principal components. This transformation was intended to retain maximum variance of the original variance as possible. While it significantly reduced the computational complexity for subsequent classification tasks, the accuracy also took a slight hit.

Following dimensionality reduction, we performed classification using two different machine learning models: k-Nearest Neighbors (kNN) and Support Vector Machines (SVM).

The kNN classifier was evaluated for values of k in {5,10,15,20} on the PCA-reduced dataset. The highest overall accuracy for kNN was achieved at k=10, with a global accuracy of 85.46%. Per-class success rates varied, with Class 1 (Trouser) achieving the highest local accuracy (95.70%) and Class 6 (Shirt) achieving the lowest (58.00%).

Compared to Midterm 1 results (without PCA), where the global accuracy was 86.17% at k=4, the performance after PCA showed a slight decrease. However, the trade-off was considered acceptable given the significant reduction in feature space and computational time.

SVM classification was performed on the PCA-reduced data using both linear and RBF kernels. The linear kernel achieved an overall accuracy of 81.39%, while the RBF kernel outperformed it

with a global accuracy of 86.69%. The RBF kernel also performed better on every class compared to the linear kernel.

When comparing the two models:

SVM with RBF kernel outperformed kNN on the PCA-reduced dataset, achieving both higher global accuracy and higher per-class success rates for most classes. While kNN's accuracy was slightly less accurate, it still maintained strong performance with a much simpler model structure and no need for model training.

The application of PCA overall helped to:

- Reduce dimensionality by over 93% (from 784 to 50 dimensions)
- Maintain good classification accuracy, with small reductions in accuracy compared to the original full-dimensional results
- Significantly improve computational efficiency during model training and testing phases

The slight decrease in classification performance after PCA is expected due to minor information loss during dimensionality reduction. However, the balance between computational speed and accuracy makes PCA an effective preprocessing step, especially for high-dimensional datasets like Fashion MNIST.

In conclusion, SVM with RBF kernel on the PCA-reduced dataset provided the best overall performance in this study. The combination of PCA followed by SVM classification is recommended for situations for maintaining high classification accuracy while minimizing computational resources is critical.

DATA604 - Midterm 2

Fashion MNIST Dataset

The dataset is taken from Zalando Research's github repository.

Link: https://github.com/zalandoresearch

In [2]:
```
# import required libraries
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
import random
```

In [3]:
```
# function to unzip data files.
# the function is taken from Zalando Research's github repository.
import os
import gzip

def load_mnist(path, kind='train'):

    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels-idx1-ubyte.gz'
                               % kind)
    images_path = os.path.join(path,
                               '%s-images-idx3-ubyte.gz'
                               % kind)

    with gzip.open(labels_path, 'rb') as lbpath:
        labels = np.frombuffer(lbpath.read(), dtype=np.uint8,
                               offset=8)

    with gzip.open(images_path, 'rb') as imgpath:
        images = np.frombuffer(imgpath.read(), dtype=np.uint8,
                               offset=16).reshape(len(labels), 784)

    return images, labels
```

In [4]:
```
# load train data
path = 'data/'
```

```
X_train, y_train = load_mnist(path, kind='train')
X_train.shape
```

Out[4]: (60000, 784)

In [5]:
```
# load test data
X_test, y_test = load_mnist(path, kind='t10k')
X_test.shape
```

Out[5]: (10000, 784)

In [6]:
```
# Standardizing the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Q1

In [8]:
```
# apply PCA
pca = PCA(n_components=X_train.shape[1])
pca.fit(X_train)

# Principal component matrix
# Shape: 784x784
P = pca.components_.T
P.shape
```

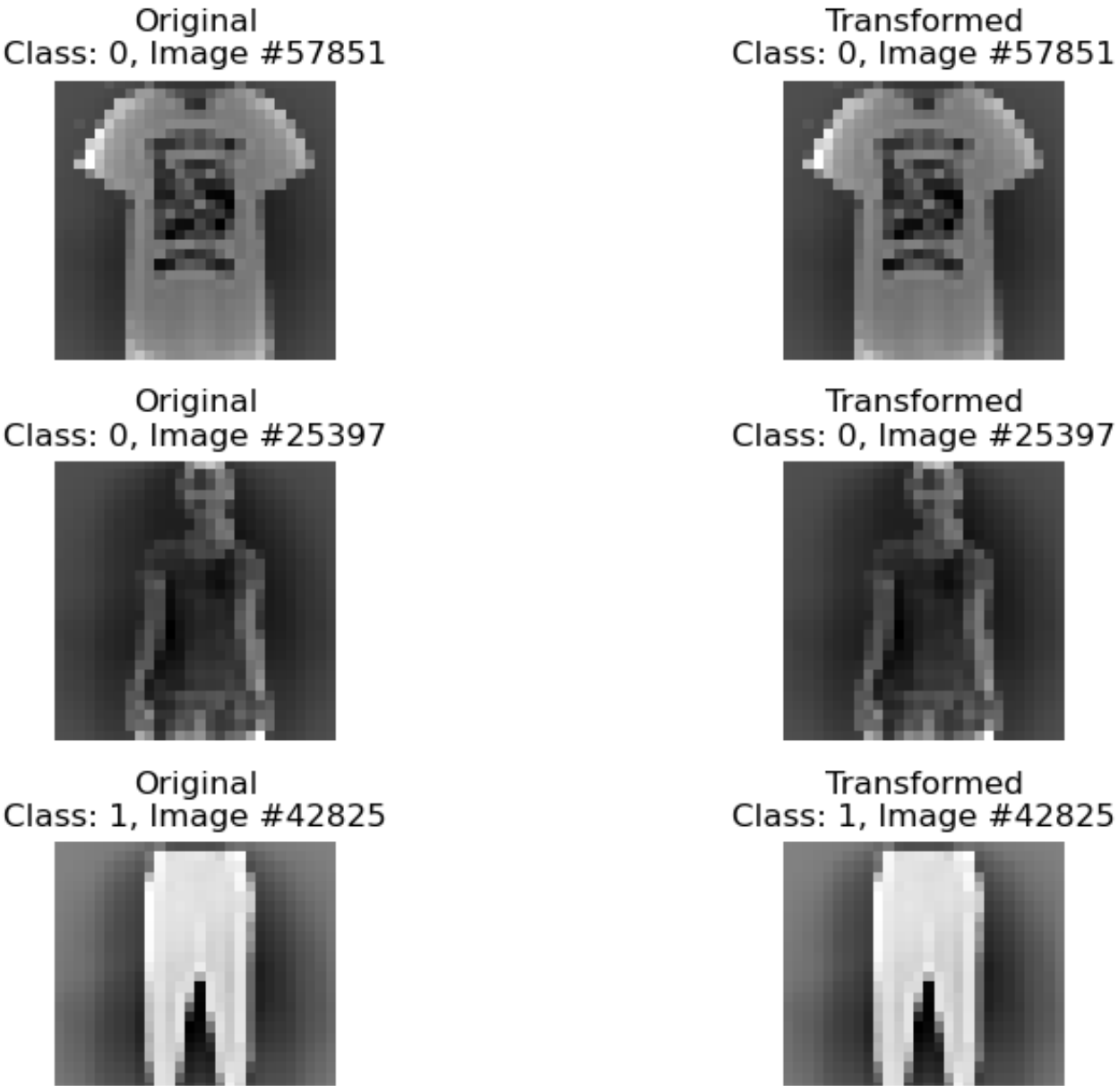
Out[8]: (784, 784)

Q1. b

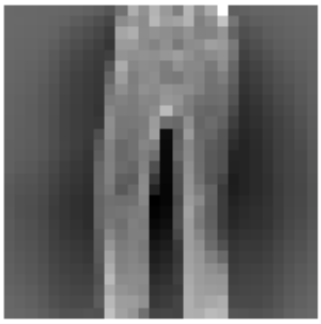
In [10]:
```
# pick 2 samples per class
p_idx = []
for label in range(10):
    idx = np.where(y_train.flatten() == label)[0]
    random_idx = random.sample(list(idx), 2)
    p_idx.extend(random_idx)
```

In [11]:
```
# Plotting the images of each class and using principal components
fig, axes = plt.subplots(20, 2, figsize=(10, 40))

for i, img_idx in enumerate(p_idx):
    x_orig = X_train[img_idx]

    # Display Original Image
```

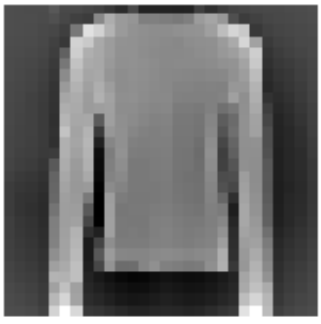
```
    axes[i, 0].imshow(x_orig.reshape(28,28), cmap='gray')
    axes[i, 0].axis('off')
    axes[i, 0].set_title(f'Original\nClass: {y_train[img_idx]}, Image #{img_

    # Transformed Image T(x) = Px
    x_transformed = P.T @ x_orig
    # Reconstruct back to pixel space
    x_reconstructed = P @ x_transformed

    # Display Reconstructed Image
    axes[i, 1].imshow(x_reconstructed.reshape(28,28), cmap='gray')
    axes[i, 1].axis('off')
    axes[i, 1].set_title(f'Transformed\nClass: {y_train[img_idx]}, Image #{i

plt.tight_layout()
plt.show()
```



Original
Class: 1, Image #58135

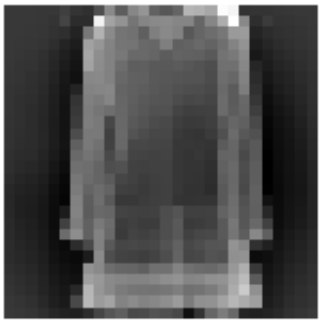


Original
Class: 2, Image #48442



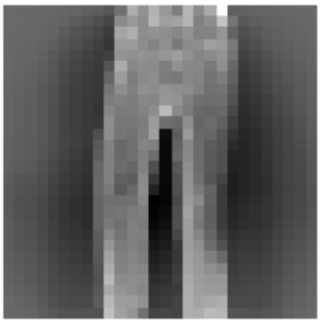
Original
Class: 2, Image #24702

Original
Class: 3, Image #9958

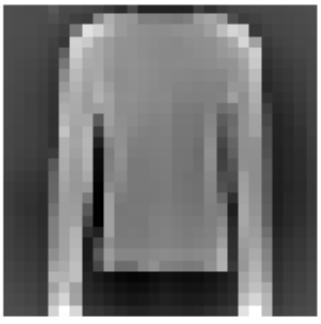


Original
Class: 3, Image #19989

Transformed
Class: 1, Image #58135



Transformed
Class: 2, Image #48442

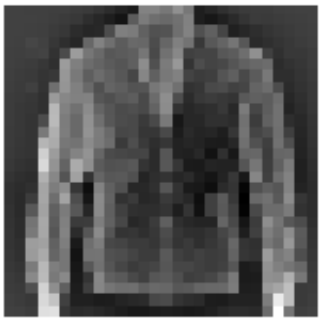


Transformed
Class: 2, Image #24702

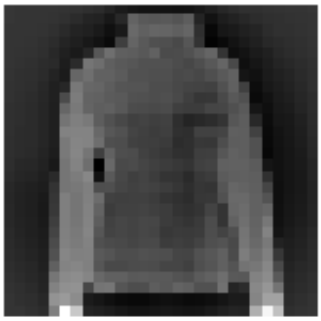
Transformed
Class: 3, Image #9958

Transformed
Class: 3, Image #19989

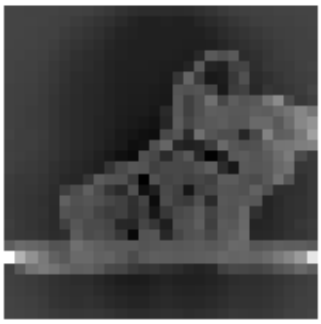
Original
Class: 4, Image #54858



Original
Class: 4, Image #6055

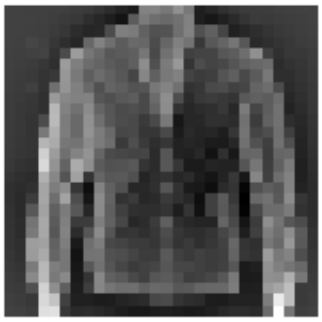


Original
Class: 5, Image #12135

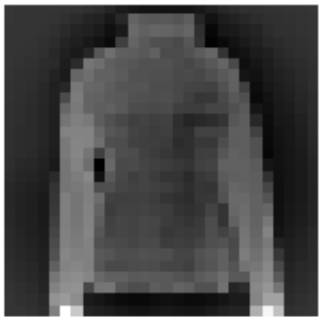


Original
Class: 5, Image #35320

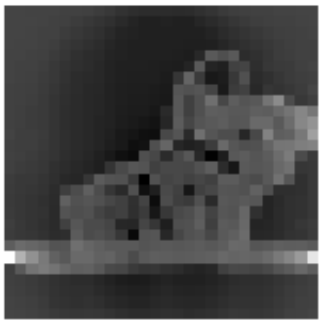
Transformed
Class: 4, Image #54858



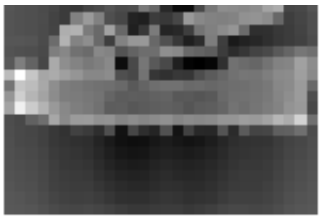
Transformed
Class: 4, Image #6055



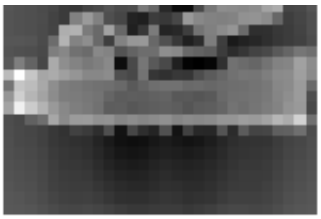
Transformed
Class: 5, Image #12135



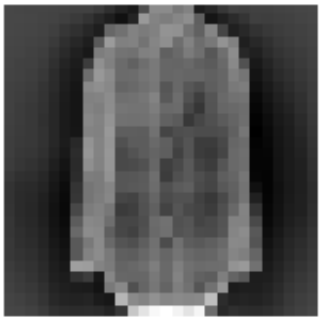
Transformed
Class: 5, Image #35320



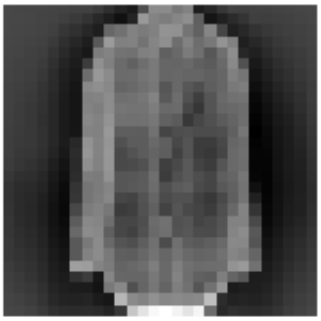
Original
Class: 6, Image #12471



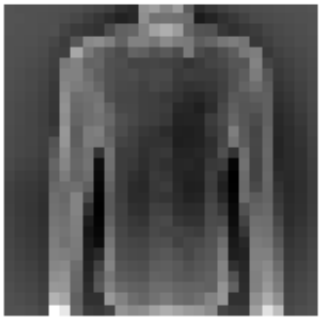
Transformed
Class: 6, Image #12471



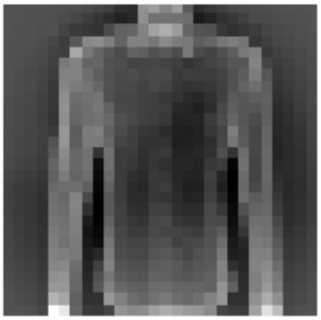
Original
Class: 6, Image #51488



Transformed
Class: 6, Image #51488



Original
Class: 7, Image #32527

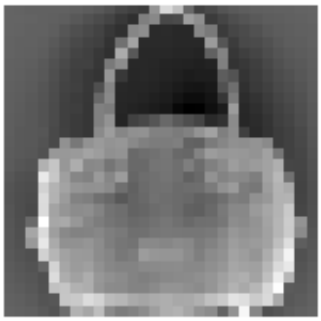


Transformed
Class: 7, Image #32527

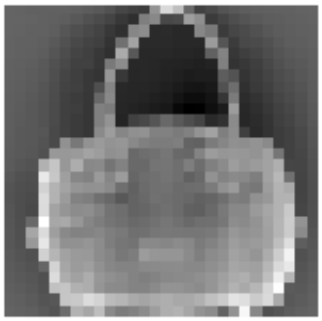
Original
Class: 7, Image #31066

Transformed
Class: 7, Image #31066

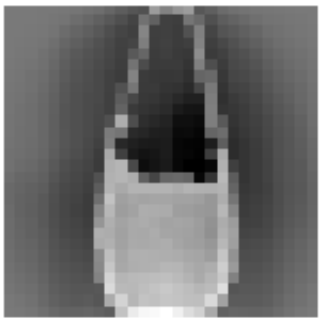
Original
Class: 8, Image #59220



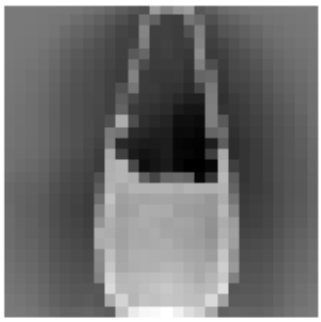
Transformed
Class: 8, Image #59220



Original
Class: 8, Image #54712



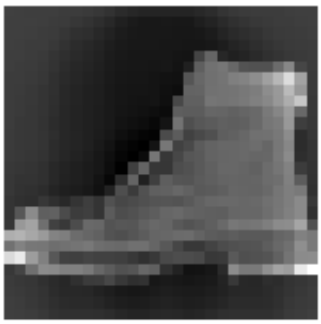
Transformed
Class: 8, Image #54712



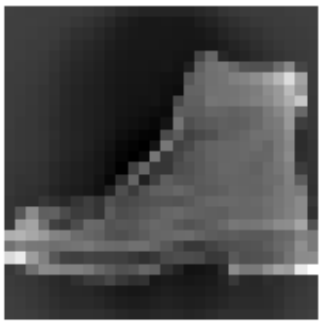
Original
Class: 9, Image #11047

Transformed
Class: 9, Image #11047

Original
Class: 9, Image #59237

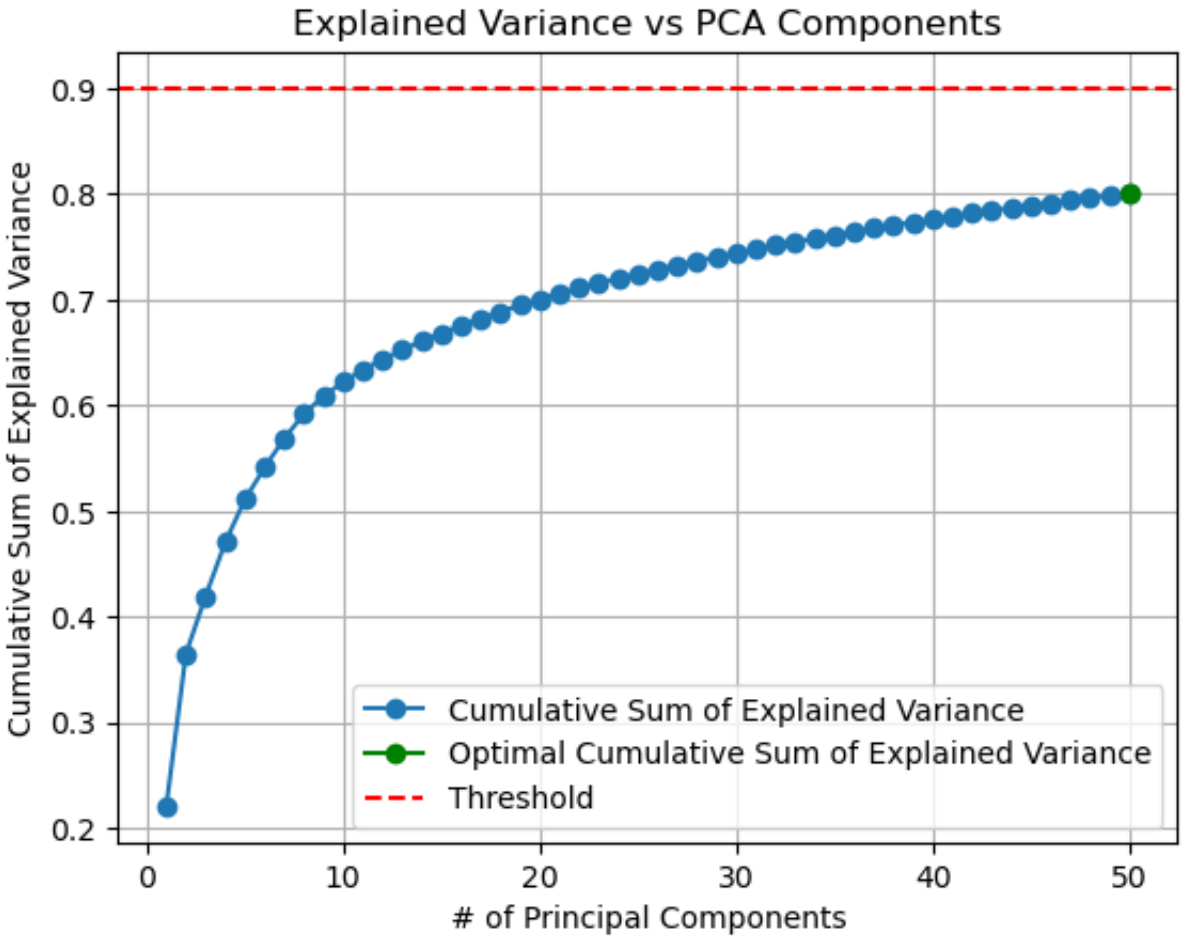


Transformed
Class: 9, Image #59237



Q1. c.

In [13]:
```
# pick optimal number of principal components between 1 and 50
# set threshold of features captured at 90%
threshold = 0.90
cumulative_var = []
max_var = 0
n_optimal = 1
for i in range(1,51):
    c_var = sum(pca.explained_variance_ratio_[:i])
    cumulative_var.append(c_var)
    if c_var>max_var and max_var<=threshold:
        cutoff_var = c_var
        n_optimal = i

plt.plot(range(1,51),cumulative_var, marker='o', label='Cumulative Sum of Ex
plt.plot(n_optimal,cutoff_var, marker='o', color = 'g', label='Optimal Cumul
plt.axhline(y = threshold, linestyle ='--', color = 'r', label='Threshold')
plt.xlabel("# of Principal Components")
plt.ylabel("Cumulative Sum of Explained Variance")
plt.title("Explained Variance vs PCA Components")
plt.legend()
plt.grid(True)
plt.show()
```



The curve starts to flatten after n_components = 10. This indicates that additional components do not contribute much. Even at 50 principal components, the cumulative explained variance is lesser than 90%. Therefore, I choose 50 principal components as the optimal number for dimensionality reduction. This choice reduces dimensionality by more than 93% (from 784 to 50). It is suitable to keep more than 90% of the data, but we are in a constraint to choose the n_components in range(1,50). Choosing more components would increase computational cost without substantial gains in explained variance.

In [15]:
```
# print cumulative sum at N_optimal
print(f"Cumulative Sum of Explained Ratio = {cutoff_var:.4f} when n_componer
```

```
Cumulative Sum of Explained Ratio = 0.8007 when n_components = 50
```

Q2

In [17]:
```
# using PCA with optimal components
pca = PCA(n_components=n_optimal)
```

```
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [18]:
```
# Run kNN and find best k value
K = [5, 10, 15, 20]
global_acc = []

# train KNN for each value of K
for k in K:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_pca, y_train)
    y_pred = knn.predict(X_test_pca)
    acc =accuracy_score(y_test, y_pred)
    global_acc.append(acc)

    print(f"\nResults for k={k}")
    print("Overall Accuracy:", acc)
    print("Success Rates per Class:")
    print(classification_report(y_test, y_pred, digits=4, zero_division=0))
```

```
Results for k=5
Overall Accuracy: 0.8506
Success Rates per Class:
              precision    recall  f1-score   support

           0     0.7710    0.8180    0.7938      1000
           1     0.9857    0.9620    0.9737      1000
           2     0.7531    0.7810    0.7668      1000
           3     0.8788    0.8630    0.8708      1000
           4     0.7425    0.7700    0.7560      1000
           5     0.9664    0.8920    0.9277      1000
           6     0.6336    0.5810    0.6062      1000
           7     0.8924    0.9370    0.9141      1000
           8     0.9724    0.9500    0.9611      1000
           9     0.9154    0.9520    0.9333      1000

    accuracy                         0.8506     10000
   macro avg     0.8511    0.8506    0.8504     10000
weighted avg     0.8511    0.8506    0.8504     10000


Results for k=10
Overall Accuracy: 0.8546
Success Rates per Class:
              precision    recall  f1-score   support

           0     0.7812    0.8460    0.8123      1000
           1     0.9876    0.9570    0.9721      1000
           2     0.7727    0.7920    0.7822      1000
           3     0.8695    0.8660    0.8677      1000
```

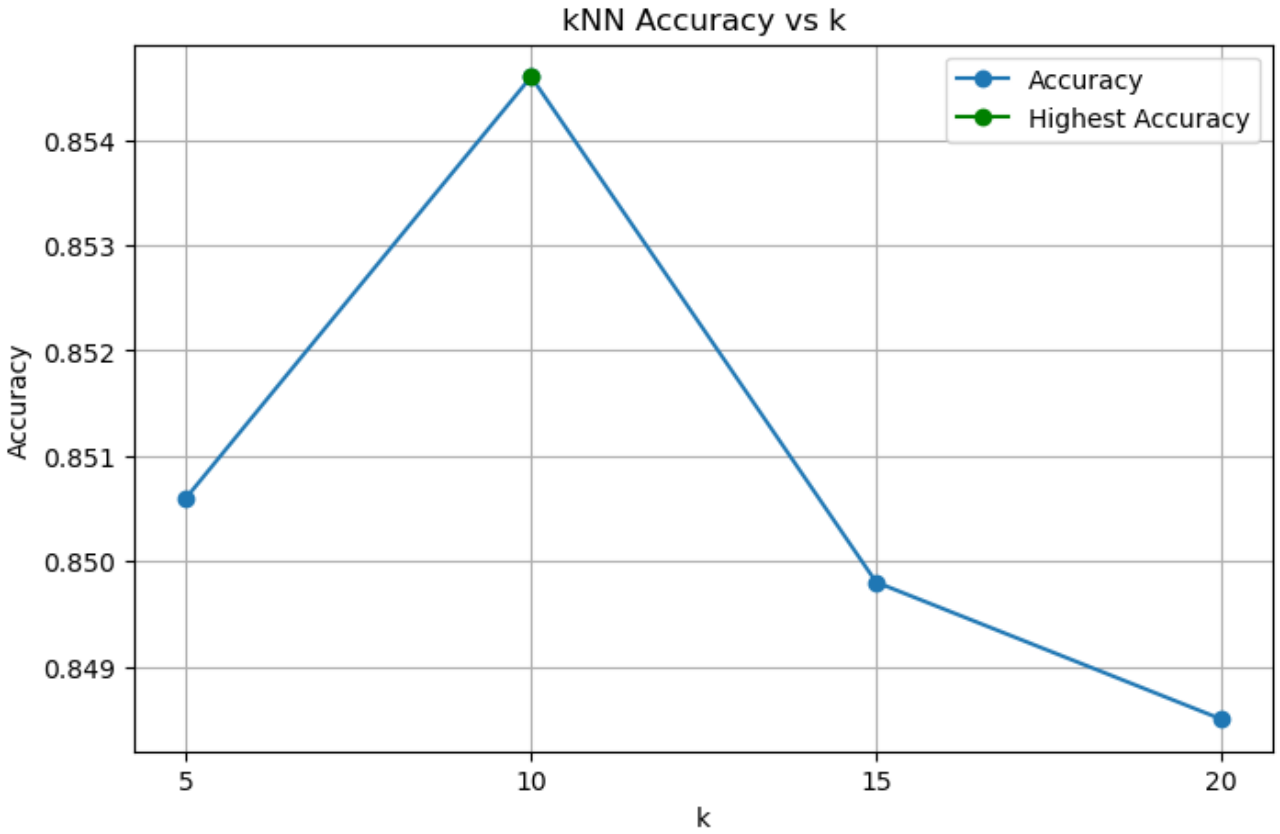
```
              4     0.7543    0.7830    0.7684      1000
              5     0.9632    0.8900    0.9252      1000
              6     0.6480    0.5800    0.6121      1000
              7     0.8913    0.9350    0.9126      1000
              8     0.9653    0.9470    0.9561      1000
              9     0.9135    0.9500    0.9314      1000

    accuracy                          0.8546     10000
   macro avg     0.8547    0.8546    0.8540     10000
weighted avg     0.8547    0.8546    0.8540     10000


Results for k=15
Overall Accuracy: 0.8498
Success Rates per Class:
              precision    recall  f1-score   support

              0     0.7865    0.8250    0.8053      1000
              1     0.9876    0.9540    0.9705      1000
              2     0.7774    0.7720    0.7747      1000
              3     0.8732    0.8680    0.8706      1000
              4     0.7493    0.7860    0.7672      1000
              5     0.9636    0.8730    0.9161      1000
              6     0.6265    0.5920    0.6087      1000
              7     0.8875    0.9310    0.9087      1000
              8     0.9545    0.9450    0.9497      1000
              9     0.8990    0.9520    0.9247      1000

    accuracy                          0.8498     10000
   macro avg     0.8505    0.8498    0.8496     10000
weighted avg     0.8505    0.8498    0.8496     10000


Results for k=20
Overall Accuracy: 0.8485
Success Rates per Class:
              precision    recall  f1-score   support

              0     0.7907    0.8350    0.8123      1000
              1     0.9896    0.9530    0.9710      1000
              2     0.7787    0.7670    0.7728      1000
              3     0.8656    0.8760    0.8708      1000
              4     0.7375    0.7840    0.7601      1000
              5     0.9593    0.8710    0.9130      1000
              6     0.6315    0.5810    0.6052      1000
              7     0.8863    0.9280    0.9067      1000
              8     0.9544    0.9410    0.9476      1000
              9     0.8953    0.9490    0.9214      1000

    accuracy                          0.8485     10000
```

```
   macro avg     0.8489    0.8485    0.8481     10000
weighted avg     0.8489    0.8485    0.8481     10000
```

In [19]:
```python
# find highest accuracy
best_idx = np.argmax(global_acc)
best_k = K[best_idx]
best_acc = global_acc[best_idx]
print(f"Highest Accuracy is {best_acc:.4f} at k={best_k}")
```

```
Highest Accuracy is 0.8546 at k=10
```

In [20]:
```python
# plot of accuracy for different values ok k
plt.figure(figsize=(8,5))
plt.plot(K,global_acc, marker='o', label='Accuracy')
plt.plot(best_k, best_acc, marker='o', color='g',label='Highest Accuracy')
plt.xlabel("k")
plt.xticks(K)
plt.ylabel("Accuracy")
plt.title("kNN Accuracy vs k")
plt.legend()
plt.grid(True)
plt.show()
```



Q3

In [22]:
```
# Run Linear SVM on the pca transformed dataset
from sklearn.svm import LinearSVC

svm = LinearSVC(dual=False, max_iter=10000)
svm.fit(X_train_pca, y_train)
y_pred = svm.predict(X_test_pca)
linear_svm_acc = accuracy_score(y_test, y_pred)

print("\n Results for Linear SVM:")
print("Overall Accuracy: ", linear_svm_acc)
print("Success Rates per Class:")
print(classification_report(y_test, y_pred, digits=4, zero_division=0))
```

```
 Results for Linear SVM:
Overall Accuracy:  0.8139
Success Rates per Class:
              precision    recall  f1-score   support

           0     0.7587    0.8080    0.7826      1000
           1     0.9655    0.9520    0.9587      1000
           2     0.7119    0.6770    0.6940      1000
           3     0.7840    0.8600    0.8202      1000
           4     0.6711    0.7650    0.7150      1000
           5     0.9059    0.8950    0.9004      1000
           6     0.5805    0.4110    0.4813      1000
           7     0.8886    0.9010    0.8947      1000
           8     0.9106    0.9370    0.9236      1000
           9     0.9129    0.9330    0.9228      1000

    accuracy                         0.8139     10000
   macro avg     0.8090    0.8139    0.8093     10000
weighted avg     0.8090    0.8139    0.8093     10000
```

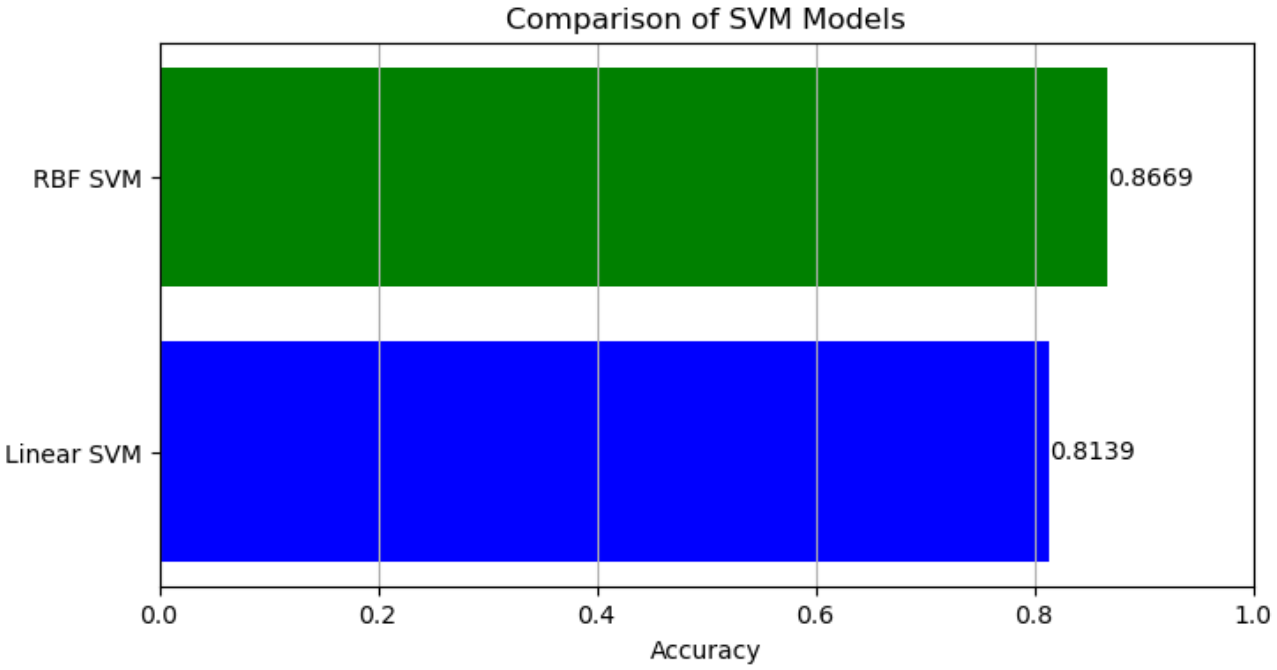
In [23]:
```
# Run SVM with RBF kernel on the pca transformed dataset

from sklearn.svm import SVC

# Train SVM using One-vs-Rest
svm = SVC(kernel='rbf', decision_function_shape='ovr')
svm.fit(X_train_pca, y_train)
y_pred = svm.predict(X_test_pca)
rbf_svm_acc = accuracy_score(y_test, y_pred)

print("\n Results for SVM using RBF kernel:")
print("Overall Accuracy: ", rbf_svm_acc)
print("Success Rates per Class:")
```

```
print(classification_report(y_test, y_pred, digits=4, zero_division=0))
```

```
 Results for SVM using RBF kernel:
Overall Accuracy:  0.8669
Success Rates per Class:
              precision    recall  f1-score   support

           0     0.8144    0.8380    0.8260      1000
           1     0.9917    0.9560    0.9735      1000
           2     0.7838    0.7720    0.7778      1000
           3     0.8523    0.8890    0.8703      1000
           4     0.7816    0.7980    0.7897      1000
           5     0.9616    0.9260    0.9435      1000
           6     0.6788    0.6360    0.6567      1000
           7     0.9082    0.9400    0.9238      1000
           8     0.9605    0.9730    0.9667      1000
           9     0.9317    0.9410    0.9363      1000

    accuracy                         0.8669     10000
   macro avg     0.8665    0.8669    0.8664     10000
weighted avg     0.8665    0.8669    0.8664     10000
```

In [25]:

```
# Plot the accuracy the SVM Models
# Data
models = ['Linear SVM', 'RBF SVM']
svm_accuracies = [linear_svm_acc, rbf_svm_acc]

# Determine colors: green for highest, blue for others
colors = ['blue', 'blue']
max_idx = np.argmax(svm_accuracies)
colors[max_idx] = 'green'

# Plot
plt.figure(figsize=(8,4))
plt.barh(models, svm_accuracies, color=colors)
plt.xlabel('Accuracy')
plt.title('Comparison of SVM Models')

# Show the accuracy value next to bars
for index, value in enumerate(svm_accuracies):
    plt.text(value, index, f'{value:.4f}', va='center')

plt.xlim(0, 1)
plt.grid(axis='x')
plt.show()
```


Comparison of SVM Models
RBF SVM
Linear SVM
0.8669
0.8139
0.0
0.2
0.4
0.6
0.8
1.0
Accuracy