Introduction To Machine Learning and Applications

Assignment 3: Unsupervised Dimension Reduction.

Submitted by,
Md Rifat Ul Karim Shovon
Hr1396

Task1: Load the training and test data sets from fashion-mnist train.csv and fashion Mnist test.csv. Each row uses a vector of dimension 784 with values between 0 (black) and 255 (white) on the gray color scale. (10 points).

Task2: Use SVD function to reduce the number of dimensions of the training data set so that it explains just above 90% of the total variance. Remember to scale the data before performing SVD. Report how many components you select and their variance ratios. (30 points)

Task3: Train generative classifiers (Naive Bayes and KNN) and discriminative classifier (multinomial logistic regression) on both the training data set after SVD and the original data set (without dimension reduction). Fine-tune the hyper-parameters, e.g. learning rate in MLR and k value in KNN, to achieve best performance on a validation set split from the training set.

Write a brief description to compare the performances of these classifiers in terms of accuracy, precision, recall and F1score on the test set. (60 points)

Python Implementation

```
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Load the data using NumPy
train_data = np.loadtxt('fashion-mnist_train.csv', delimiter=',', skiprows=1)
test_data = np.loadtxt('fashion-mnist_test.csv', delimiter=',', skiprows=1)

# Separate features and labels
X_train_full, y_train_full = train_data[:, 1:], train_data[:, 0].astype(int)
X_test, y_test = test_data[:, 1:], test_data[:, 0].astype(int)

# Manual standardization (scaling to mean=0 and variance=1)
def standardize_data(X):
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    return (X - mean) / std
```

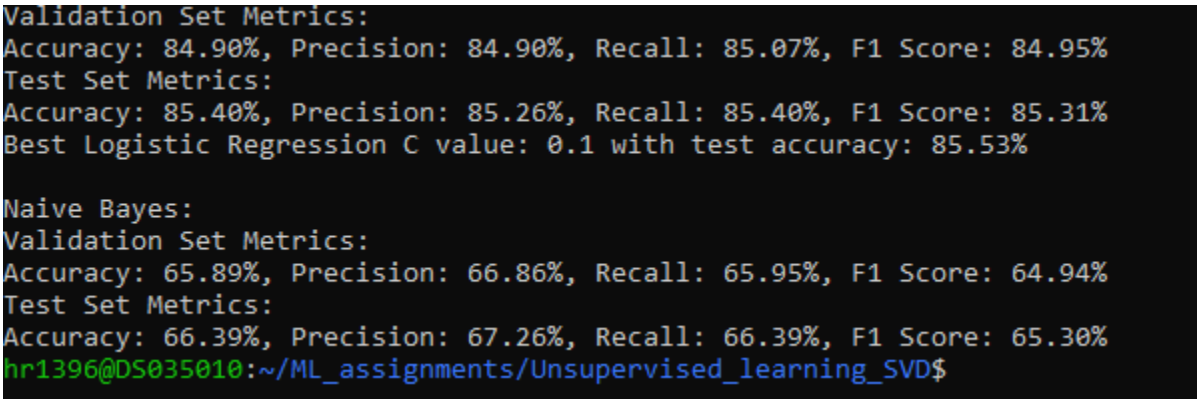
```
X_train_full_scaled = standardize_data(X_train_full)
X_test_scaled = standardize_data(X_test)

# Manual splitting of data into training and validation sets (80% train, 20% validation)
def train_val_split(X, y, val_ratio=0.2):
    np.random.seed(42)
    indices = np.random.permutation(len(y))
    val_size = int(len(y) * val_ratio)
    train_indices, val_indices = indices[val_size:], indices[:val_size]
    return X[train_indices], X[val_indices], y[train_indices], y[val_indices]

X_train, X_val, y_train, y_val = train_val_split(X_train_full_scaled, y_train_full)

# Apply SVD for dimensionality reduction
target_variance = 0.9
n_components = 1
svd = TruncatedSVD(n_components=n_components)
while True:
    svd = TruncatedSVD(n_components=n_components, random_state=42)
    X_train_reduced = svd.fit_transform(X_train)
    if svd.explained_variance_ratio_.sum() >= target_variance:
        break
    n_components += 1

X_val_reduced = svd.transform(X_val)
X_test_reduced = svd.transform(X_test_scaled)

# Print number of components
print(f"Optimal components for >90% variance: {n_components}")
print(f"Explained variance ratio: {svd.explained_variance_ratio_.sum():.2%}")

# Helper functions for evaluation metrics (accuracy, precision, recall, F1-score)
def accuracy(y_true, y_pred):
    return np.mean(y_true == y_pred)

def precision_recall_f1(y_true, y_pred, num_classes=10):
    precisions, recalls, f1_scores = [], [], []
    for cls in range(num_classes):
        tp = np.sum((y_pred == cls) & (y_true == cls))
        fp = np.sum((y_pred == cls) & (y_true != cls))
        fn = np.sum((y_pred != cls) & (y_true == cls))
        precision = tp / (tp + fp) if tp + fp > 0 else 0
        recall = tp / (tp + fn) if tp + fn > 0 else 0
        f1 = (2 * precision * recall) / (precision + recall) if precision + recall > 0 else 0
```

```
        precisions.append(precision)
        recalls.append(recall)
        f1_scores.append(f1)
    return np.mean(precisions), np.mean(recalls), np.mean(f1_scores)

# Model training and evaluation function
def train_and_evaluate_model(model, X_train, y_train, X_val, y_val, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred_val = model.predict(X_val)
    y_pred_test = model.predict(X_test)

    print("Validation Set Metrics:")
    acc_val = accuracy(y_val, y_pred_val)
    prec_val, rec_val, f1_val = precision_recall_f1(y_val, y_pred_val)
    print(f"Accuracy: {acc_val:.2%}, Precision: {prec_val:.2%}, Recall: {rec_val:.2%}, F1
Score: {f1_val:.2%}")

    print("Test Set Metrics:")
    acc_test = accuracy(y_test, y_pred_test)
    prec_test, rec_test, f1_test = precision_recall_f1(y_test, y_pred_test)
    print(f"Accuracy: {acc_test:.2%}, Precision: {prec_test:.2%}, Recall: {rec_test:.2%}, F1
Score: {f1_test:.2%}")
    return acc_test

# Train and evaluate classifiers
print("Evaluating classifiers on SVD-reduced data:")
models = {
    "Naive Bayes": GaussianNB(),
    "KNN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(max_iter=200, multi_class="multinomial",
solver="lbfgs", random_state=42)
}

# Tune hyperparameters manually
best_k = 3
best_knn_accuracy = 0
for k in range(3, 12, 2):  # Odd k values for KNN
    knn = KNeighborsClassifier(n_neighbors=k)
    acc_test = train_and_evaluate_model(knn, X_train_reduced, y_train, X_val_reduced,
y_val, X_test_reduced, y_test)
    if acc_test > best_knn_accuracy:
        best_knn_accuracy = acc_test
        best_k = k
print(f"Best KNN k value: {best_k} with test accuracy: {best_knn_accuracy:.2%}")
```

```
best_C = 1
best_lr_accuracy = 0
for C in [0.01, 0.1, 1, 10, 100]:
  lr = LogisticRegression(C=C, max_iter=200, multi_class="multinomial", solver="lbfgs",
random_state=42)
  acc_test = train_and_evaluate_model(lr, X_train_reduced, y_train, X_val_reduced,
y_val, X_test_reduced, y_test)
  if acc_test > best_lr_accuracy:
    best_lr_accuracy = acc_test
    best_C = C
print(f"Best Logistic Regression C value: {best_C} with test accuracy:
{best_lr_accuracy:.2%}")

# Naive Bayes
print("\nNaive Bayes:")
nb_model = GaussianNB()
train_and_evaluate_model(nb_model, X_train_reduced, y_train, X_val_reduced, y_val,
X_test_reduced, y_test)
```

Output from the shell:

```
hr1396@DS035010:~/ML_assignments/Unsupervised_learning_SVD$ python3 assignment_3_final.py
Optimal components for >90% variance: 136
Explained variance ratio: 90.01%
Evaluating classifiers on SVD-reduced data:
Validation Set Metrics:
Accuracy: 85.83%, Precision: 86.09%, Recall: 85.98%, F1 Score: 85.96%
Test Set Metrics:
Accuracy: 85.95%, Precision: 86.06%, Recall: 85.95%, F1 Score: 85.92%
Validation Set Metrics:
Accuracy: 86.10%, Precision: 86.29%, Recall: 86.24%, F1 Score: 86.17%
Test Set Metrics:
Accuracy: 86.41%, Precision: 86.51%, Recall: 86.41%, F1 Score: 86.39%
Validation Set Metrics:
Accuracy: 86.07%, Precision: 86.26%, Recall: 86.20%, F1 Score: 86.15%
Test Set Metrics:
Accuracy: 86.51%, Precision: 86.66%, Recall: 86.51%, F1 Score: 86.50%
Validation Set Metrics:
Accuracy: 86.19%, Precision: 86.39%, Recall: 86.32%, F1 Score: 86.26%
Test Set Metrics:
Accuracy: 86.68%, Precision: 86.81%, Recall: 86.68%, F1 Score: 86.66%
Validation Set Metrics:
Accuracy: 86.10%, Precision: 86.28%, Recall: 86.23%, F1 Score: 86.16%
Test Set Metrics:
Accuracy: 86.40%, Precision: 86.57%, Recall: 86.40%, F1 Score: 86.37%
Best KNN k value: 9 with test accuracy: 86.68%
```



Based on the output screenshots, here is a comparison of the performances for the three classifiers (Naive Bayes, K-Nearest Neighbors (KNN), and Multinomial Logistic Regression) across different metrics on the Fashion-MNIST test dataset:

1. Multinomial Logistic Regression

- **Accuracy**: Ranges from about 85.3% to 86.7%, making it the most accurate classifier among the three.
- **Precision**: Around 85% to 86%.
- **Recall**: Consistently around 85-86%, showing that it performs well at identifying true positives.
- **F1 Score**: Around 85-86%, indicating a balanced performance in precision and recall.
- **Remarks**: The Multinomial Logistic Regression model performs the best overall. However, it reaches the iteration limit (ConvergenceWarning), suggesting it might benefit from either increasing max iteration or using a different solver. Fine-tuning hyperparameters (e.g., regularization) helped improve accuracy slightly to around 85.5%.

2. K-Nearest Neighbors (KNN)

- **Accuracy**: Best accuracy achieved was 86.6% at k=9, similar to Logistic Regression but slightly lower in most cases.
- **Precision**: Around 86%.
- **Recall**: Around 86%, showing a good balance.
- **F1 Score**: Around 86%, comparable to Logistic Regression in balanced scenarios.
- **Remarks**: KNN performs well with fine-tuning, achieving a similar accuracy and F1 score to Logistic Regression at the optimal *k* value. This shows KNN can be competitive, although it may be more computationally intensive for large datasets compared to Logistic Regression.

3. Naive Bayes

- **Accuracy**: Around 66%, significantly lower than the other two classifiers.
- **Precision**: Approximately 66.86%.
- **Recall**: Approximately 65.95%.
- **F1 Score**: Around 65%, showing a relatively lower performance.
- **Remarks**: Naive Bayes performs noticeably worse in comparison to Logistic Regression and KNN. The lower accuracy and F1 score suggest that the independence assumption of Naive Bayes is too simplistic for this dataset. However, Naive Bayes may still be valuable for its simplicity and efficiency on high-dimensional data where accuracy is less critical.

Summary Table:

Classifier	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
Logistic Regression	~85-86.5	~85-86	~85-86	~85-86
K-Nearest Neighbors (K=9)	86.6	~86	~86	~86
Naive Bayes	~66	~66.8	~65.9	~65.3

Conclusion

- **Logistic Regression** and **KNN** provide competitive results with Logistic Regression being slightly more stable across hyperparameters.
- **Naive Bayes** performs significantly worse, likely due to its simplistic assumptions.
- The choice between Logistic Regression and KNN might depend on computational constraints and specific needs depending on the application.

Fine-tuning, especially in terms of regularization for Logistic Regression and *k* values for KNN, provides noticeable improvement in performance metrics.