Python for Computer Science and Data Science 2 (CSE 3652)

MAJOR ASSIGNMENT-3: MACHINE LEARNING- CLASSIFICATION, REGRESSION AND CLUSTERING

Overview

Imagine you've been hired by *FashionX*, a fast-growing online fashion retailer struggling to organize its expanding product catalog. The company needs an automated system to classify fashion items into categories like T-shirts, Dresses, Shoes, etc., based on images.

As a data scientist, your task is to develop a machine learning model to classify 28x28 pixel images from the Fashion *MNIST* dataset into 10 fashion categories. You'll apply algorithms like K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) for classification. Additionally, you'll use t-SNE to visualize the data, uncovering patterns and clusters in the high-dimensional space.

This project will help FashionX scale its operations by efficiently automating product categorization, making it easier for customers to find what they're looking for.

Tasks

Task 1: Data Preprocessing

- Download the Fashion MNIST dataset using TensorFlow. You can load it directly using the following code:

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
# Load the dataset
(train_images, train_labels), (test_images, test_labels) = \
fashion_mnist.load_data()
```

- The dataset consists of 60,000 training images and 10,000 test images of fashion products, with 10 distinct categories. Each image is 28x28 pixels.
- Perform the following preprocessing steps:
  1. Normalize the images (values should be between 0 and 1).
  2. Flatten the 28x28 pixel images into 1D arrays (784 pixels per image).
  3. Handle any missing or incorrect values (if any).
- Split the dataset into training and test sets.

Task 2: K-Nearest Neighbors (KNN) Classification

- Implement the K-Nearest Neighbors (KNN) algorithm using the preprocessed dataset.
- Experiment with different values of k (e.g., $k = 3$, $k = 5$, $k = 7$) to see how the model performance changes.
- Evaluate the model performance using accuracy on the test dataset.
- Provide a comparison of different k values and the impact on accuracy.

Task 3: Support Vector Machine (SVM) Classification

- Train a Support Vector Machine (SVM) classifier on the same preprocessed dataset.
- Experiment with different kernels (linear, polynomial, radial basis function) and hyperparameters such as C.
- Evaluate the model performance using accuracy on the test dataset.
- Compare the SVM performance with that of the KNN model.

Task 4: Data Visualization with t-SNE

- Use the t-SNE technique to reduce the dimensionality of the data from 784 features to 2 or 3 dimensions.

- Visualize the 2D or 3D representation of the Fashion MNIST dataset and observe the clustering of different fashion categories.
- Analyze the plot to identify how well the categories are separated, and discuss the results.

Task 5: Model Evaluation and Reporting

- Evaluate the performance of both the KNN and SVM models using accuracy, precision, recall, F1-score, and confusion matrix.
- Discuss which model performs better and why, based on the evaluation metrics.
- Write a report summarizing the approach used in each task, the results obtained, and insights derived from the visualizations.

Code:

```
# Task 1: Data Preprocessing
import tensorflow as tf
from sklearn.model_selection import train_test_split
import numpy as np
# Load dataset
(train_images, train_labels), (test_images, test_labels) =
      tf.keras.datasets.fashion_mnist.load_data()
# Normalize
train_images = train_images / 255.0
test_images = test_images / 255.0
# Flatten
X_train = train_images.reshape(-1, 784)
X_test = test_images.reshape(-1, 784)
# Check for NaNs
assert not np.isnan(X_train).any() and not np.isnan(X_test).any()
# Labels are already in numeric form (0-9)
y_train, y_test = train_labels, test_labels



# Task 2: K-Nearest Neighbors (KNN) Classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
for k in [3, 5, 7]:
      knn = KNeighborsClassifier(n_neighbors=k)
      knn.fit(X_train, y_train)
      y_pred = knn.predict(X_test)
      acc = accuracy_score(y_test, y_pred)
      print(f"K={k}, Accuracy={acc:.4f}")



# Task 3: Support Vector Machine (SVM) Classification
from sklearn.svm import SVC
kernels = ['linear', 'poly', 'rbf']
```

```
for kernel in kernels:
      svm = SVC(kernel=kernel, C=1.0)
      svm.fit(X_train[:10000], y_train[:10000]) # limit for performance
      y_pred = svm.predict(X_test[:2000])
      acc = accuracy_score(y_test[:2000], y_pred)
      print(f"SVM Kernel={kernel}, Accuracy={acc:.4f}")



# Task 4: Data Visualization with t-SNE
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
X_embedded = tsne.fit_transform(X_test[:1000])
y_sample = y_test[:1000]
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=y_sample,
      cmap='tab10', s=10)
plt.title('t-SNE visualization of Fashion MNIST')
plt.colorbar(scatter, ticks=range(10))
plt.show()



# Task 5: Model Evaluation and Reporting
from sklearn.metrics import classification_report, confusion_matrix
# Use best KNN (e.g., K=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn_preds = knn.predict(X_test)
# Use best SVM (e.g., RBF kernel)
svm = SVC(kernel='rbf')
svm.fit(X_train[:10000], y_train[:10000])
svm_preds = svm.predict(X_test[:2000])
# Evaluation
print("KNN Classification Report:")
print(classification_report(y_test, knn_preds))
print("SVM Classification Report:")
print(classification_report(y_test[:2000], svm_preds))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, knn_preds))
print("SVM Confusion Matrix:\n", confusion_matrix(y_test[:2000], svm_preds))
```

Output:

```
K=3, Accuracy=0.8541
K=5, Accuracy=0.8554
K=7, Accuracy=0.8540
SVM Kernel=linear, Accuracy=0.8305
SVM Kernel=poly, Accuracy=0.8280
SVM Kernel=rbf, Accuracy=0.8600
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.85      0.81      1000
           1       0.99      0.97      0.98      1000
           2       0.73      0.82      0.77      1000
           3       0.90      0.86      0.88      1000
           4       0.79      0.77      0.78      1000
           5       0.99      0.82      0.90      1000
           6       0.66      0.57      0.61      1000
           7       0.88      0.96      0.92      1000
           8       0.97      0.95      0.96      1000
           9       0.90      0.97      0.93      1000

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.85     10000
weighted avg       0.86      0.86      0.85     10000

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.80      0.82       200
           1       0.97      0.95      0.96       203
           2       0.77      0.80      0.79       214
           3       0.80      0.88      0.84       190
           4       0.80      0.77      0.79       219
           5       0.93      0.92      0.93       195
           6       0.66      0.66      0.66       197
           7       0.90      0.94      0.92       200
           8       0.98      0.95      0.97       194
           9       0.97      0.93      0.95       188

    accuracy                           0.86      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.86      0.86      2000
```

```
KNN Confusion Matrix:
 [[855   1  17  16   3   1 100   1   6   0]
 [  8 968   4  12   4   0   3   0   1   0]
 [ 24   2 819  11  75   0  69   0   0   0]
 [ 41   8  15 860  39   0  34   0   3   0]
 [  2   1 126  26 773   0  71   0   1   0]
 [  1   0   0   0   0 822   5  96   1  75]
 [176   1 132  23  80   0 575   0  13   0]
 [  0   0   0   0   0   3   0 961   0  36]
 [  2   0  10   4   7   0  16   7 953   1]
 [  0   0   0   0   0   2   1  29   0 968]]
SVM Confusion Matrix:
 [[159   1   2  14   1   0  23   0   0   0]
 [  0 192   0  11   0   0   0   0   0   0]
 [  2   0 172   2  22   0  16   0   0   0]
 [  5   3   1 168   5   0   7   0   1   0]
 [  0   0  25   6 169   0  18   0   1   0]
 [  0   0   0   0   0 180   0  13   0   2]
 [ 22   0  22   6  14   0 131   0   2   0]
 [  0   0   0   0   0   7   0 189   0   4]
 [  0   1   0   2   0   2   4   0 185   0]
 [  0   0   0   0   0   4   0   9   0 175]]
```