Applied Machine Learning

Homework 12

Anand Kamble
Department of Scientific Computing
Florida State University

Classification Using Linear Classifier and PPCA

In this report, we perform classification on a subset of the ImageNet dataset using a linear classifier with cross-entropy loss and a Probabilistic PCA (PPCA)-based classifier. The dataset consists of 10 classes, each containing approximately 1300 observations with 640 features. We standardize the data, train both classifiers, and evaluate their performance on both training and test sets.

Approach and Implementation

Data Loading and Preprocessing

We load the training and test data from the provided `.mat` files and concatenate them to form the feature matrices `X_train` and `X_test`, and the label vectors `y_train` and `y_test`:

```python
import numpy as np
import scipy.io
import os
from sklearn.preprocessing import StandardScaler
# Directories containing the data
trainDataDir = 'features_640'  # Replace with your path
testDataDir = 'features_val_640'  # Replace with your path

# Get list of files and sort them to ensure consistent class labels
trainFiles:list[str] = sorted(os.listdir(trainDataDir))
testFiles:list[str] = sorted(os.listdir(testDataDir))

# Initialize lists to hold data and labels
xTrainList:list = []
yTrainList:list = []

xTestList:list = []
yTestList:list = []

# Load training data
print("Loading training data...")
for idx, filename in enumerate(tqdm(trainFiles)):
    filepath: str = os.path.join(trainDataDir, filename)
    matContents:dict = scipy.io.loadmat(filepath)
    # Extract features
    data_keys:list = [key for key in matContents.keys() if not key.startswith('__')]
    features = matContents['feature']
    # Assign labels based on the index (from 0 to 9)
    labels = np.full(features.shape[0], idx)
    xTrainList.append(features)
    yTrainList.append(labels)

# Concatenate all training data
xTrain = np.vstack(xTrainList)
yTrain = np.concatenate(yTrainList)
```

a) Training a Linear Classifier with Cross-Entropy Loss

We train a linear classifier using a simple neural network with one linear layer and cross-entropy loss. We use stochastic gradient descent (SGD) with a learning rate scheduler:

```
import torch
import torch.nn as nn
import torch.optim as optim
# Convert data to PyTorch tensors
xTrainTensor:torch.Tensor = torch.from_numpy(xTrainScaled).float()
yTrainTensor:torch.Tensor = torch.from_numpy(yTrain).long()
xTestTensor:torch.Tensor = torch.from_numpy(xTestScaled).float()
yTestTensor:torch.Tensor = torch.from_numpy(yTest).long()
# Part a) Train a linear classifier using cross-entropy loss (softmax loss)
class LinearClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(LinearClassifier, self).__init__()
        self.linear = nn.Linear(input_dim, num_classes)

    def forward(self, x):
        return self.linear(x)

input_dim = xTrainTensor.shape[1]
num_classes = 10

model = LinearClassifier(input_dim, num_classes)
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
# Training loop
numEpochs = 30
batchSize = 64
trainDataset = torch.utils.data.TensorDataset(xTrainTensor, yTrainTensor)
trainLoader = torch.utils.data.DataLoader(trainDataset, batch_size=batchSize, shuffle=True)

for epoch in range(numEpochs):
    model.train()
    runningLoss = 0.0
    for inputs, labels in trainLoader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        runningLoss += loss.item() * inputs.size(0)

    scheduler.step()
```

Evaluation of the Linear Classifier

We evaluate the trained linear classifier on both the training and test sets:

```
model.eval()
with torch.no_grad():
    outputs = model(xTrainTensor)
    _, yTrainPred = torch.max(outputs, 1)
    trainAccuracy:int | float | bool = (yTrainPred == yTrainTensor).float().mean().item()
    trainMisclassificationError:int | float = 1 - trainAccuracy

# Evaluate on test data
with torch.no_grad():
    outputs = model(xTestTensor)
    _, yTestPred = torch.max(outputs, 1)
    testAccuracy:int | float | bool = (yTestPred == yTestTensor).float().mean().item()
    testMisclassificationError:int | float = 1 - testAccuracy
```

Results

Metric	Training Set	Test Set
Misclassification Error	0.0007	0.0200

Table 1: Misclassification Errors of the Linear Classifier

b) Training PPCA Models for Each Class

We train a PPCA model for each class using $q = 20$ principal components and report the estimated σ_k^2 for each class:

```
q = 20  # Number of principal components
muList:list = []
wList:list = []
sigma2List:list = []
for classIdx in tqdm(range(num_classes)):
    X_class = xTrainScaled[yTrain == classIdx]
    N_k, D = X_class.shape
    # Compute the mean
    mu_k = np.mean(X_class, axis=0)
    # Center the data
    X_centered = X_class - mu_k
    # Perform SVD
    U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
    # Select top q components
    W_k = Vt[:q].T  # D x q
    # Estimate sigma^2
    residualVariances = (S[q:] ** 2) / (N_k - 1)
    sigma2K = np.mean(residualVariances)
    muList.append(mu_k)
    wList.append(W_k)
    sigma2List.append(sigma2K)
    print(f"Class {classIdx}: sigma^2 = {sigma2K:.4f}")
```

Estimated σ_k^2 Values

Class	σ_k^2
0	0.1937
1	0.2336
2	0.2760
3	0.2351
4	0.3771
5	0.2265
6	0.3050
7	0.5595
8	0.4016
9	0.4436

Table 2: Estimated σ_k^2 for Each Class

c) Classification Using Mahalanobis Distance

We classify each sample by computing the Mahalanobis distance to each class's PPCA model and assigning it to the class with the smallest distance:

```
def compute_mahalanobis_distance(x, mu_k, W_k, sigma2_k):
    diff:np.ndarray = x - mu_k  # D-dimensional
    M_inv:np.ndarray = np.linalg.inv(np.eye(W_k.shape[1]) + (W_k.T @ W_k) / sigma2_k)
    term1:np.ndarray = (diff @ diff) / sigma2_k
    term2:np.ndarray = (diff @ W_k @ M_inv @ W_k.T @ diff) / (sigma2_k ** 2)
    dist:np.ndarray = term1 - term2
    return dist

def classify_ppca(X, muList, wList, sigma2List):
    N:int = X.shape[0]
    num_classes:int = len(muList)
    y_pred:np.ndarray = np.zeros(N, dtype=int)

    for i in tqdm(range(N)):
        x:np.ndarray = X[i]
        distances: np.ndarray = np.zeros(num_classes)
        for k in range(num_classes):
            dist = compute_mahalanobis_distance(x, muList[k], wList[k], sigma2List[k])
            distances[k] = dist
        y_pred[i] = np.argmin(distances)
    return y_pred
```

Results

Metric	Training Set	Test Set
Misclassification Error	0.0797	0.1060

Table 3: Misclassification Errors of the PPCA-Based Classifier

Appendix: Full Code

```
#%%
import numpy as np
from numpy._typing._array_like import NDArray
import scipy.io
import os
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.optim as optim

# Set random seeds for reproducibility
np.random.seed(42)
torch.manual_seed(42)

# Directories containing the data
trainDataDir = 'features_640'  # Replace with your path
testDataDir = 'features_val_640'  # Replace with your path

# Get list of files and sort them to ensure consistent class labels
trainFiles:list[str] = sorted(os.listdir(trainDataDir))
testFiles:list[str] = sorted(os.listdir(testDataDir))

# Initialize lists to hold data and labels
xTrainList:list = []
yTrainList:list = []

xTestList:list = []
yTestList:list = []

# Load training data
print("Loading training data...")
for idx, filename in enumerate(tqdm(trainFiles)):
    filepath: str = os.path.join(trainDataDir, filename)
    matContents:dict = scipy.io.loadmat(filepath)

    # Extract features
    data_keys:list = [key for key in matContents.keys() if not key.startswith('__')]

    features = matContents['feature']

    # Assign labels based on the index (from 0 to 9)
    labels = np.full(features.shape[0], idx)

    xTrainList.append(features)
    yTrainList.append(labels)

# Concatenate all training data
xTrain = np.vstack(xTrainList)
yTrain = np.concatenate(yTrainList)

# Load test data
print("Loading test data...")
for idx, filename in enumerate(tqdm(testFiles)):
    filepath = os.path.join(testDataDir, filename)
```

```
    matContents = scipy.io.loadmat(filepath)
    # Extract features
    data_keys:list = [key for key in matContents.keys() if not key.startswith('__')]
    # Try to extract features
    features:np.ndarray = matContents['feature']
    # Assign labels based on the index (from 0 to 9)
    labels:np.ndarray = np.full(features.shape[0], idx)
    xTestList.append(features)
    yTestList.append(labels)

# Concatenate all test data
xTest:np.ndarray = np.vstack(xTestList)
yTest:np.ndarray = np.concatenate(yTestList)

# Standardize the data
print("Standardizing data...")
scaler = StandardScaler()
xTrainScaled:np.ndarray = scaler.fit_transform(xTrain)
xTestScaled :np.ndarray= scaler.transform(xTest)  # Use the same mean and std

# Convert data to PyTorch tensors
xTrainTensor:torch.Tensor = torch.from_numpy(xTrainScaled).float()
yTrainTensor:torch.Tensor = torch.from_numpy(yTrain).long()

xTestTensor:torch.Tensor = torch.from_numpy(xTestScaled).float()
yTestTensor:torch.Tensor = torch.from_numpy(yTest).long()

# Part a) Train a linear classifier using cross-entropy loss (softmax loss)
print("Training linear classifier with cross-entropy loss...")

class LinearClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(LinearClassifier, self).__init__()
        self.linear = nn.Linear(input_dim, num_classes)

    def forward(self, x):
        return self.linear(x)

input_dim = xTrainTensor.shape[1]
num_classes = 10

model = LinearClassifier(input_dim, num_classes)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

# Training loop
numEpochs = 30
batchSize = 64

trainDataset = torch.utils.data.TensorDataset(xTrainTensor, yTrainTensor)
trainLoader = torch.utils.data.DataLoader(trainDataset, batch_size=batchSize,
↪  shuffle=True)

for epoch in range(numEpochs):
    model.train()
```

```
    runningLoss = 0.0
    for inputs, labels in trainLoader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        runningLoss += loss.item() * inputs.size(0)

    scheduler.step()
    epochLoss = runningLoss / len(trainDataset)
    print(f"Epoch {epoch+1}/{numEpochs}, Loss: {epochLoss:.4f}")

# Evaluate on training data
model.eval()
with torch.no_grad():
    outputs = model(xTrainTensor)
    _, yTrainPred = torch.max(outputs, 1)
    trainAccuracy:int | float | bool = (yTrainPred ==
    ↪  yTrainTensor).float().mean().item()
    trainMisclassificationError:int | float = 1 - trainAccuracy

# Evaluate on test data
with torch.no_grad():
    outputs = model(xTestTensor)
    _, yTestPred = torch.max(outputs, 1)
    testAccuracy:int | float | bool = (yTestPred ==
    ↪  yTestTensor).float().mean().item()
    testMisclassificationError:int | float = 1 - testAccuracy

print(f"Training misclassification error: {trainMisclassificationError:.4f}")
print(f"Test misclassification error: {testMisclassificationError:.4f}")

# Part b) Train PPCA models for each class
print("Training PPCA models for each class...")
q = 20  # Number of principal components
muList:list = []
wList:list = []
sigma2List:list = []

for classIdx in tqdm(range(num_classes)):
    X_class = xTrainScaled[yTrain == classIdx]
    N_k, D = X_class.shape

    # Compute the mean
    mu_k = np.mean(X_class, axis=0)

    # Center the data
    X_centered = X_class - mu_k

    # Perform SVD
    U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)

    # Select top q components
    W_k = Vt[:q].T  # D x q

    # Estimate sigma^2
    residualVariances = (S[q:] ** 2) / (N_k - 1)
```

```
    sigma2K = np.mean(residualVariances)

    muList.append(mu_k)
    wList.append(W_k)
    sigma2List.append(sigma2K)

    print(f"Class {classIdx}: sigma^2 = {sigma2K:.4f}")

# Part c) Classification using Mahalanobis distance
print("Classifying data using PPCA-based classifier...")

def compute_mahalanobis_distance(x, mu_k, W_k, sigma2_k):
    diff:np.ndarray = x - mu_k  # D-dimensional
    M_inv:np.ndarray = np.linalg.inv(np.eye(W_k.shape[1]) + (W_k.T @ W_k) /
    ↪  sigma2_k)
    term1:np.ndarray = (diff @ diff) / sigma2_k
    term2:np.ndarray = (diff @ W_k @ M_inv @ W_k.T @ diff) / (sigma2_k ** 2)
    dist:np.ndarray = term1 - term2
    return dist

def classify_ppca(X, muList, wList, sigma2List):
    N:int = X.shape[0]
    num_classes:int = len(muList)
    y_pred:np.ndarray = np.zeros(N, dtype=int)

    for i in tqdm(range(N)):
        x:np.ndarray = X[i]
        distances: np.ndarray = np.zeros(num_classes)
        for k in range(num_classes):
            dist = compute_mahalanobis_distance(x, muList[k], wList[k],
            ↪  sigma2List[k])
            distances[k] = dist
        y_pred[i] = np.argmin(distances)
    return y_pred

# Classify training data
print("Classifying training data...")
y_train_pred_ppca = classify_ppca(xTrainScaled, muList, wList, sigma2List)
train_accuracy_ppca = accuracy_score(yTrain, y_train_pred_ppca)
train_misclassification_error_ppca = 1 - train_accuracy_ppca
print(f"PPCA-based classifier training misclassification error:
↪  {train_misclassification_error_ppca:.4f}")

# Classify test data
print("Classifying test data...")
y_test_pred_ppca = classify_ppca(xTestScaled, muList, wList, sigma2List)
test_accuracy_ppca = accuracy_score(yTest, y_test_pred_ppca)
test_misclassification_error_ppca = 1 - test_accuracy_ppca
print(f"PPCA-based classifier test misclassification error:
↪  {test_misclassification_error_ppca:.4f}")
```

Appendix: Terminal Output

```
(homeworks) amk23j@class107 ~/Documents/FSU-assignments/Machine Learning/HW12
↪  (main)$ python main.py
Loading training data...
100%|████████████████████████████████████████████████████████████| 10/10
↪  [00:00<00:00, 365.53it/s]
Loading test data...
100%|████████████████████████████████████████████████████████████| 10/10
↪  [00:00<00:00, 1743.99it/s]
Standardizing data...
Training linear classifier with cross-entropy loss...
Epoch 1/30, Loss: 0.0727
Epoch 2/30, Loss: 0.0247
Epoch 3/30, Loss: 0.0184
Epoch 4/30, Loss: 0.0150
Epoch 5/30, Loss: 0.0126
Epoch 6/30, Loss: 0.0112
Epoch 7/30, Loss: 0.0097
Epoch 8/30, Loss: 0.0087
Epoch 9/30, Loss: 0.0080
Epoch 10/30, Loss: 0.0073
Epoch 11/30, Loss: 0.0063
Epoch 12/30, Loss: 0.0062
Epoch 13/30, Loss: 0.0062
Epoch 14/30, Loss: 0.0061
Epoch 15/30, Loss: 0.0061
Epoch 16/30, Loss: 0.0060
Epoch 17/30, Loss: 0.0060
Epoch 18/30, Loss: 0.0060
Epoch 19/30, Loss: 0.0059
Epoch 20/30, Loss: 0.0059
Epoch 21/30, Loss: 0.0058
Epoch 22/30, Loss: 0.0058
Epoch 23/30, Loss: 0.0058
Epoch 24/30, Loss: 0.0058
Epoch 25/30, Loss: 0.0058
Epoch 26/30, Loss: 0.0058
Epoch 27/30, Loss: 0.0058
Epoch 28/30, Loss: 0.0058
Epoch 29/30, Loss: 0.0058
Epoch 30/30, Loss: 0.0057
Training misclassification error: 0.0007
Test misclassification error: 0.0200
Training PPCA models for each class...
  0%|                                          | 0/10 [00:00<?, ?it/s]Class 0:
  ↪  sigma^2 = 0.1937
 10%|███                                | 1/10 [00:00<00:01,  5.64it/s]Class 1:
 ↪  sigma^2 = 0.2336
 20%|██████                                | 2/10 [00:00<00:01,  5.96it/s]Class 2:
 ↪  sigma^2 = 0.2760
 30%|█████████                                | 3/10 [00:00<00:01,  6.06it/s]Class
 ↪  3: sigma^2 = 0.2351
 40%|████████████                                | 4/10 [00:00<00:01,
 ↪  5.98it/s]Class 4: sigma^2 = 0.3771
 50%|███████████████                                | 5/10 [00:00<00:00,
 ↪  5.74it/s]Class 5: sigma^2 = 0.2265
```

```
 60%|                                          | 6/10 [00:01<00:00,
 ↪  5.82it/s]Class 6: sigma^2 = 0.3050
 70%|                                            | 7/10 [00:01<00:00,
 ↪  5.87it/s]Class 7: sigma^2 = 0.5595
 80%|                                             | 8/10
 ↪  [00:01<00:00,  5.89it/s]Class 8: sigma^2 = 0.4016
 90%|                                              | 9/10
 ↪  [00:01<00:00,  6.17it/s]Class 9: sigma^2 = 0.4436
100%|                                             | 10/10
↪  [00:01<00:00,  5.88it/s]
Classifying data using PPCA-based classifier...
Classifying training data...
100%|                                            | 12855/12855 [00:06<00:00,
↪  2007.04it/s]
PPCA-based classifier training misclassification error: 0.0797
Classifying test data...
100%|                                            | 500/500
↪  [00:00<00:00, 1937.13it/s]
PPCA-based classifier test misclassification error: 0.1060
```