Advanced Machine Learning Models for Data Analytics

Assignment **3**
Professor: **Dr. Ruchika Arora**
Submitted By: **Rohit kumar** (Mtech_AI-**25901334**)

1. Problem

The minority class constitutes **0.5%** of the dataset.
Explain **SMOTE**, provide **pseudocode** to generate synthetic minority samples using **k-Nearest Neighbors**, and explain how **Tomek Links** can be applied after SMOTE and why this combination is effective. Also provide working Python code.

1.1 Explanation of SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a widely used oversampling method designed to address extreme class imbalance. Instead of duplicating rare samples, SMOTE **creates new synthetic minority points** by interpolating between a minority sample and one of its k-nearest minority neighbors.

For a given minority point xxx:

1. Identify its *k* nearest neighbors among the minority class.

2. Randomly pick one neighbor xnnx_{nn}xnn.

3. Generate a synthetic point along the line segment between them:

xsynthetic=x+λ(xnn−x)x_{\text{synthetic}} = x + \lambda (x_{nn} - x)xsynthetic=x+λ(xnn−x)

where λ∈[0,1)\lambda \in [0,1)λ∈[0,1) is a random number.

This expands the decision region of the minority class and increases variety without overfitting to exact duplicates.

1.2 SMOTE Pseudocode (k-NN based synthetic sample generation)

```
Input:
  M = {x1, x2, ..., xn}   // minority samples
  k: number of nearest neighbors
  m: number of synthetic samples to generate per minority sample

Output:
  S = synthetic samples

For each xi in M:
    neighbors = k nearest neighbors of xi within M
    Repeat m times:
        Randomly select neighbor x_nn from neighbors
        Generate λ = random number in [0,1)
        x_syn = xi + λ * (x_nn - xi)
        Append x_syn to S

Return S
```

This pseudocode ensures new points lie inside the minority manifold, improving classifier generalization.

1.3 Tomek Links After SMOTE

A **Tomek Link** is a pair of samples from opposite classes that are each other's nearest neighbors.
If two such points exist:

- The pair lies very close to the decision boundary.
- Often, the **majority class** member is noisy or overlapping.

Using Tomek Links after SMOTE

Workflow:

1. **Apply SMOTE** → minority class is expanded.
2. **Identify Tomek Links** in the SMOTE-augmented dataset.
3. **Remove majority-class samples** that participate in Tomek Links.

Why SMOTE + Tomek Links Helps

- SMOTE may create synthetic minority points that lie too close to majority samples.
- Tomek Links remove ambiguous majority samples at the boundary.
- The combination:
    - cleans class overlap,
    - sharpens class boundaries,
    - reduces noise,
    - improves classifier performance on the minority class.

1.4 Python Code for SMOTE + Tomek Links (Working Example)

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import TomekLinks
from sklearn.model_selection import train_test_split

# 1. Create imbalanced dataset (0.5% minority)
X, y = make_classification(
    n_samples=20000,
    n_features=20,
    n_informative=6,
    weights=[0.995, 0.005],
    random_state=42
)

print("Original:", Counter(y))

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2, random_state=1
)

print("Train before SMOTE:", Counter(y_train))

# 2. Apply SMOTE
sm = SMOTE(k_neighbors=5, sampling_strategy=0.10,
random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
print("After SMOTE:", Counter(y_res))

# 3. Apply Tomek Links to clean noisy majority points
```

```
tl = TomekLinks()
X_clean, y_clean = tl.fit_resample(X_res, y_res)
print("After SMOTE + Tomek:", Counter(y_clean))

# 4. PCA visualization (optional)
pca = PCA(n_components=2)
X_train_vis = pca.fit_transform(X_train)
X_clean_vis = pca.transform(X_clean)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.title("Original Training Data (PCA)")
plt.scatter(X_train_vis[y_train==0,0],
X_train_vis[y_train==0,1], s=6)
plt.scatter(X_train_vis[y_train==1,0],
X_train_vis[y_train==1,1], s=16, c='red')

plt.subplot(1, 2, 2)
plt.title("After SMOTE + Tomek (PCA)")
plt.scatter(X_clean_vis[y_clean==0,0],
X_clean_vis[y_clean==0,1], s=6)
plt.scatter(X_clean_vis[y_clean==1,0],
X_clean_vis[y_clean==1,1], s=16, c='red')

plt.show()
```

This code demonstrates full workflow:
imbalanced dataset → SMOTE → Tomek Links → cleaned dataset.

2. Problem

With **100 labeled medical images**, propose a **transfer-learning plan** using a pretrained CNN backbone. Explain which layers to freeze, when to fine-tune, and which augmentations maximize generalization.

2.1 Transfer Learning Strategy for 100 Medical Images

Since the dataset is extremely small, transfer learning is essential. A pretrained CNN (e.g., ResNet50, DenseNet121, EfficientNet-B0) provides strong general visual features that can be adapted to medical imaging.

Stage 1: Feature Extraction (Freeze Backbone)

- Use ImageNet-pretrained backbone.
- **Freeze all convolutional layers** because low-level features (edges, textures) transfer well.
- Replace final classifier with:
    - Global Average Pooling
    - Dense (256) + ReLU
    - Dropout (0.5)
    - Dense (#Classes) + Softmax

Train only this new head for 20–50 epochs using a learning rate around **1e-3**.

Purpose:
Learn dataset-specific decision boundaries without disturbing the powerful pretrained filters.

Stage 2: Fine-Tuning (Unfreeze Higher Layers)

After stabilizing the classifier head:

- Unfreeze the **last 1–2 convolutional blocks** of the backbone (e.g., ResNet's *layer4*, optionally *layer3*).
- Keep early layers frozen (they encode general visual features).
- Train with a very small learning rate (e.g., backbone LR = **1e-5**, head LR = **1e-4**).

Purpose:
Adapt deeper semantic features to the medical domain (textures, lesions, anatomical structures).

Use early stopping to prevent overfitting.

2.2 Recommended Data Augmentation

Since only 100 images are available, strong augmentations help generalization.

Safe Augmentations

- Random rotations (±15–30°)
- Random zoom (0.8–1.2 scale)
- Horizontal/vertical flips
- Random brightness/contrast/gamma
- Small shifts/translation
- Gaussian noise
- Random cropping + resize

Biomedical-specific Augmentations

- Elastic deformations (useful for microscopy/histopathology)
- Color jitter or stain augmentation (for pathology images)

Advanced Augmentations

- **Mixup** and **CutMix** (strong regularization)
- **Test-Time Augmentation (TTA)** for inference

2.3 Example Transfer Learning Workflow

1. Load pretrained CNN backbone
2. Freeze backbone
3. Train classifier head (moderate LR)
4. Unfreeze last few layers
5. Fine-tune with reduced LR
6. Use strong data augmentation
7. Evaluate using k-fold cross-validation
8. Apply TTA during prediction

2.4 Compact Code Skeleton (PyTorch)

```
import torch
```

```
import torchvision
from torch import nn, optim
from torchvision import transforms, models

model = models.resnet50(pretrained=True)

# Replace final layer
in_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Linear(in_features, 256),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(256, num_classes)
)

# Stage 1: Freeze backbone
for name, param in model.named_parameters():
    if "fc" not in name:
        param.requires_grad = False

# Train classifier head...

# Stage 2: Unfreeze last block
for name, param in model.named_parameters():
    if "layer4" in name or "fc" in name:
        param.requires_grad = True

# Fine-tune with low LR...
```

Final Summary

SMOTE

- Creates synthetic minority points by interpolating between minority neighbors.
- Helps expand minority class decision space.

Tomek Links

- Remove noisy / overlapping majority samples.
- Used after SMOTE to clean boundaries.