Hướng dẫn 9 Công cụ TDA cho Phân lớp Văn bản Tiếng Việt

Tổng quan

Trong **Phân tích Dữ liệu Tô-pô (TDA)**, có 9 công cụ chính được sử dụng để phân tích cấu trúc dữ liệu và áp dụng vào bài toán **"Cải tiến hiệu suất phân lớp văn bản tiếng Việt bằng cách kết hợp kỹ thuật tăng cường dữ liệu với tô-pô và học sâu"**.

Danh sách 9 Công cụ TDA Chính

1. Persistent Homology (Phép đồng điều dai dẳng)

- **Mô tả**: Theo dõi các đặc trưng tô-pô qua các tham số filtration
- **Ứng dụng**: Xác định bình luận có cấu trúc ngữ nghĩa độc đáo
- **Thư viện**: persim, Ripser

2. Vietoris-Rips Complex (Phức hợp Vietoris-Rips)

- **Mô tả**: Xây dựng phức hợp simplicial từ khoảng cách điểm
- **Ứng dụng**: Tạo cấu trúc tô-pô từ nhúng văn bản
- **Thư viện**: Gudhi

3. Wasserstein Distance (Khoảng cách Wasserstein)

- **Mô tả**: Đo lường khác biệt giữa persistence diagrams
- **Ứng dụng**: So sánh cấu trúc tô-pô giữa các bình luận
- **Thư viện**: persim

4. Persistence Images (Hình ảnh dai dẳng)

- **Mô tả**: Chuyển persistence diagrams thành ma trận hình ảnh
- **Ứng dụng**: Tích hợp với mạng neural CNN
- **Thư viện**: persim

5. Persistence Landscapes (Phong cảnh dai dẳng)

- **Mô tả**: Vector hóa persistence diagrams thành hàm số
- **Ứng dụng**: Ổn định hơn persistence images trong một số trường hợp
- **Thư viện**: Gudhi

6. Betti Numbers (Số Betti)

- **Mô tả**: Đếm số lượng đặc trưng tô-pô (β_0: cụm, β_1: lỗ, β_2: khoảng trống)
- **Ứng dụng**: Trích xuất đặc trưng số học đơn giản
- **Thư viện**: Gudhi

7. Mapper Algorithm (Thuật toán Mapper)

- **Mô tả**: Tạo đồ thị biểu diễn cụm dữ liệu
- **Ứng dụng**: Xác định cụm ngữ nghĩa thưa thớt
- **Thư viện**: Gudhi

8. Multi-dimensional Persistence (Đồng điều dai dẳng đa chiều)

- **Mô tả**: Phân tích với nhiều tham số filtration
- **Ứng dụng**: Phân tích đa chiều phức tạp
- **Thư viện**: Gudhi

9. Persistent Entropy (Entropy dai dẳng)

- **Mô tả**: Đo lường độ phức tạp persistence diagrams
- **Ứng dụng**: Ưu tiên tăng cường dữ liệu phức tạp
- **Thư viện**: persim

Cài đặt Thư viện

```bash
pip install ripser persim gudhi sentence-transformers scikit-learn matplotlib
```

Ví dụ Code Tổng hợp

1. Persistent Homology

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
from ripser import ripser
from persim import plot_diagrams
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm.", "Giá hợp lý."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tính persistence diagram
diagrams = ripser(embeddings_scaled, maxdim=1)['dgms']
plot_diagrams(diagrams, show=True)
plt.title("Persistence Diagram của Nhúng Văn Bản")
plt.show()

# Chọn bình luận để tăng cường
threshold = 0.5
diagrams_1 = diagrams[1][diagrams[1][:, 1] != np.inf]
selected_indices = [i for i, (birth, death) in enumerate(diagrams_1) if (death - birth) > threshold]
print("Bình luận được chọn để tăng cường:")
for idx in selected_indices:
    print(f"- {texts[idx]}")
```

2. Vietoris-Rips Complex

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
import gudhi as gd
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tạo Vietoris-Rips complex
rips_complex = gd.RipsComplex(points=embeddings_scaled, max_edge_length=1.0)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=1)
diag = simplex_tree.persistence()

# Chọn bình luận để tăng cường
threshold = 0.5
diag_1 = [point[1] for point in diag if point[0] == 1]
selected_texts = [texts[i] for i, (birth, death) in enumerate(diag_1) if (death - birth) > threshold]
print("Bình luận được chọn để tăng cường:")
for text in selected_texts:
    print(f"- {text}")
```

3. Wasserstein Distance

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
from ripser import ripser
from persim import wasserstein
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tính persistence diagrams
diagrams = ripser(embeddings_scaled, maxdim=1)['dgms'][1]
diagrams = diagrams[diagrams[:, 1] != np.inf]

# Tính Wasserstein distance
distances = []
for i in range(len(texts)):
    for j in range(i + 1, len(texts)):
        dist = wasserstein(diagrams, diagrams, i, j)
        distances.append((i, j, dist))

# Chọn bình luận có khoảng cách lớn
threshold = 0.5
selected_indices = [i for i, j, dist in distances if dist > threshold]
print("Bình luận được chọn để tăng cường:")
for idx in set(selected_indices):
    print(f"- {texts[idx]}")
```

4. Persistence Images

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
from ripser import ripser
from persim import PersistenceImager
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm.", "Giá hợp lý."]
labels = [1, 0, 1, 1]  # 1: tích cực, 0: tiêu cực
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa và tính persistence diagram
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)
diagrams = ripser(embeddings_scaled, maxdim=1)['dgms'][1]
diagrams = diagrams[diagrams[:, 1] != np.inf]

# Tạo persistence image
pimgr = PersistenceImager(resolution=(50, 50), pixel_size=0.1)
persistence_image = pimgr.fit_transform([diagrams])[0].flatten()

# Kết hợp đặc trưng
combined_features = np.concatenate([embeddings, np.tile(persistence_image, (len(texts), 1))], axis=1)

# Phân lớp
clf = LogisticRegression()
clf.fit(combined_features, labels)
print(f"Độ chính xác: {clf.score(combined_features, labels)}")

# Trực quan hóa persistence image
plt.imshow(pimgr.fit_transform([diagrams])[0], cmap='hot')
plt.title("Persistence Image của Nhúng Văn Bản")
plt.colorbar()
plt.show()
```

5. Persistence Landscapes

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
import gudhi as gd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm.", "Giá hợp lý."]
labels = [1, 0, 1, 1]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tính persistence diagram
rips_complex = gd.RipsComplex(points=embeddings_scaled, max_edge_length=1.0)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=1)
diag = simplex_tree.persistence()
diag_1 = [point[1] for point in diag if point[0] == 1]

# Tạo persistence landscape
landscape = gd.representations.Landscape(resolution=50)
landscape_vector = landscape.fit_transform([diag_1])[0]

# Kết hợp đặc trưng
combined_features = np.concatenate([embeddings, np.tile(landscape_vector, (len(texts), 1))], axis=1)

# Phân lớp
clf = LogisticRegression()
clf.fit(combined_features, labels)
print(f"Độ chính xác: {clf.score(combined_features, labels)}")
```

6. Betti Numbers

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
import gudhi as gd
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tính Betti numbers
rips_complex = gd.RipsComplex(points=embeddings_scaled, max_edge_length=1.0)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
diag = simplex_tree.persistence()
betti_0 = sum(1 for point in diag if point[0] == 0 and point[1][1] != float('inf'))
betti_1 = sum(1 for point in diag if point[0] == 1 and point[1][1] != float('inf'))
print(f"Betti Numbers: β₀ = {betti_0}, β₁ = {betti_1}")

# Sử dụng Betti numbers làm đặc trưng
betti_features = np.array([[betti_0, betti_1]] * len(texts))
combined_features = np.concatenate([embeddings, betti_features], axis=1)
```

7. Mapper Algorithm

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
import gudhi as gd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm.", "Giá hợp lý."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tạo Mapper
mapper = gd.mapper.MapperComplex(
    input_type="point cloud",
    colors=embeddings_scaled[:, 0],
    resolutions=[5],
    gains=[0.5],
    clustering=gd.sklearn_clustering.DBSCAN(eps=0.5, min_samples=1)
)
node_info, edge_info = mapper.fit_transform(embeddings_scaled)

# Chọn cụm thưa thớt
sparse_clusters = [i for i, node in enumerate(node_info) if len(node) < 2]
selected_texts = [texts[i] for i in sparse_clusters for i in node_info[i]]
print("Bình luận cần tăng cường:")
for text in selected_texts:
    print(f"- {text}")
```

8. Multi-dimensional Persistence

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
import gudhi as gd
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Mô phỏng multi-dimensional persistence
thresholds = [0.5, 1.0]
selected_texts = []
for threshold in thresholds:
    rips_complex = gd.RipsComplex(points=embeddings_scaled, max_edge_length=threshold)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=1)
    diag = simplex_tree.persistence(min_persistence=0.1)
    diag_1 = [point[1] for point in diag if point[0] == 1]
    persistences = [(birth, death, death - birth) for birth, death in diag_1]
    selected = [texts[i] for i, p in enumerate(persistences) if p[2] > 0.5]
    selected_texts.extend(selected)

print("Bình luận cần tăng cường (multi-parameter):")
for text in list(set(selected_texts)):
    print(f"- {text}")
```

9. Persistent Entropy

python

```python
import numpy as np
from sentence_transformers import SentenceTransformer
from ripser import ripser
from persim import pers_entropy
from sklearn.preprocessing import StandardScaler

# Dữ liệu mẫu
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!", "Món ăn ngon, phục vụ chậm.", "Giá hợp lý."]
model = SentenceTransformer('distiluse-base-multilingual-cased-v2')
embeddings = model.encode(texts)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Tính persistence diagram
diagrams = ripser(embeddings_scaled, maxdim=1)['dgms'][1]
diagrams = diagrams[diagrams[:, 1] != np.inf]

# Tính persistent entropy
entropy = pers_entropy([diagrams])[0]
print(f"Persistent Entropy: {entropy}")

# Chọn bình luận nếu entropy cao
if entropy > 0.5:  # Ngưỡng tùy chọn
    print("Bình luận cần tăng cường (entropy cao):")
    for text in texts:
        print(f"- {text}")
```

Tích hợp vào Dự án

Kết hợp với PhoBERT

python

```python
from transformers import AutoModel, AutoTokenizer
import torch

# Load PhoBERT
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
model = AutoModel.from_pretrained("vinai/phobert-base")

def get_phobert_embeddings(texts):
    encoded = tokenizer(texts, padding=True, truncation=True, return_tensors='pt')
    with torch.no_grad():
        outputs = model(**encoded)
    return outputs.last_hidden_state.mean(dim=1).numpy()

# Sử dụng với TDA
texts = ["Sản phẩm tuyệt vời!", "Dịch vụ tệ!"]
embeddings = get_phobert_embeddings(texts)
# Áp dụng các công cụ TDA như trên...
```

Tích hợp với Deep Learning

python

```python
import torch
import torch.nn as nn

class TDAEnhancedClassifier(nn.Module):
    def __init__(self, embedding_dim, tda_feature_dim, num_classes):
        super().__init__()
        self.embedding_layer = nn.Linear(embedding_dim, 512)
        self.tda_layer = nn.Linear(tda_feature_dim, 128)
        self.classifier = nn.Linear(640, num_classes)

    def forward(self, embeddings, tda_features):
        emb_out = torch.relu(self.embedding_layer(embeddings))
        tda_out = torch.relu(self.tda_layer(tda_features))
        combined = torch.cat([emb_out, tda_out], dim=1)
        return self.classifier(combined)
```

Tài liệu Tham khảo

- **Persim**: https://persim.scikit-tda.org/
- **Gudhi**: http://pinpoint.gudhi.inria.fr/
- **Ripser**: https://ripser.scikit-tda.org/
- **SentenceTransformers**: https://www.sbert.net/

Lưu ý Kỹ thuật

1. **Chuẩn hóa dữ liệu**: Luôn sử dụng StandardScaler trước khi tính toán TDA
2. **Tham số điều chỉnh**: max_edge_length, max_dimension cần được điều chỉnh theo dữ liệu
3. **Hiệu suất**: Với dữ liệu lớn, nên sử dụng batch processing
4. **Visualization**: Sử dụng matplotlib để trực quan hóa persistence diagrams

Kết luận

9 công cụ TDA trên cung cấp một bộ công cụ đầy đủ để phân tích cấu trúc tô-pô của dữ liệu văn bản tiếng Việt, giúp cải thiện hiệu suất phân lớp thông qua việc tăng cường dữ liệu thông minh và kết hợp với các mô hình học sâu.