一. (20 points) 降维

1. 基于 numpy 和 fetch_lfw_people 数据集实现主成分分析（PCA）算法，不可以调用 sklearn 库，完成下面代码并且可视化前 5 个主成分所对应的特征脸 (10 points)

```python
from sklearn.datasets import fetch_lfw_people
import numpy as np
import matplotlib.pyplot as plt

lfw_people = fetch_lfw_people(min_faces_per_person=100, resize=0.4)

X = lfw_people.data
image_shape = lfw_people.images[0].shape

def PCA1(X, n_components=5):
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean

    covariance_matrix = np.dot(X_centered.T, X_centered) / X_centered.shape[0]

    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    eigenfaces = eigenvectors[:, :n_components].T

    return eigenfaces, X_mean

eigenfaces, X_mean = PCA1(X, n_components=5)

plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(eigenfaces[i].reshape(image_shape), cmap='gray')
    plt.title(f"PC {i+1}")
    plt.axis('off')
plt.suptitle("Top 5 Principal Components (Eigenfaces)")
plt.show()
```

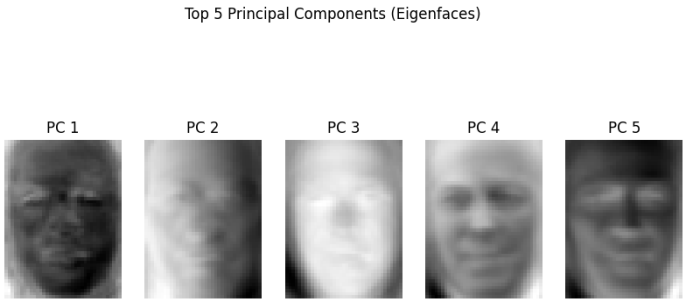


图 1: 前 5 个主成分所对应的特征脸的可视化

2. 根据局部线性嵌入 (Locally Linear Embedding, LLE) 的算法流程，尝试编写 LLE 代码，可以基于 sklearn 实现，并在瑞士卷数据集上进行实验降到 2 维空间。提交代码和展示多个在不同参数下的可视化的实验结果。请分析使用 LLE 时可能遇到哪些挑战 (10 points) [提示：瑞士卷数据集可以用 sklearn 的 make_swiss_roll(n_samples=3000, random_state=0) 生成 3000 个样本]

```python
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding
import matplotlib.pyplot as plt
import numpy as np

n_samples = 3000
X, color = make_swiss_roll(n_samples=n_samples, random_state=0)

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral, s=1)
plt.title("Original Swiss Roll Dataset")
plt.show()

params = [5, 10, 15]
plt.figure(figsize=(15, 5))

for i, n_neighbors in enumerate(params):
    lle = LocallyLinearEmbedding(n_neighbors=n_neighbors, n_components=2, random_state=0)
    X_r = lle.fit_transform(X)

    plt.subplot(1, len(params), i + 1)
    plt.scatter(X_r[:, 0], X_r[:, 1], c=color, cmap=plt.cm.Spectral, s=5)
    plt.title(f"LLE (n_neighbors={n_neighbors})")
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")

plt.suptitle("LLE on Swiss Roll with Different n_neighbors")
plt.show()
```

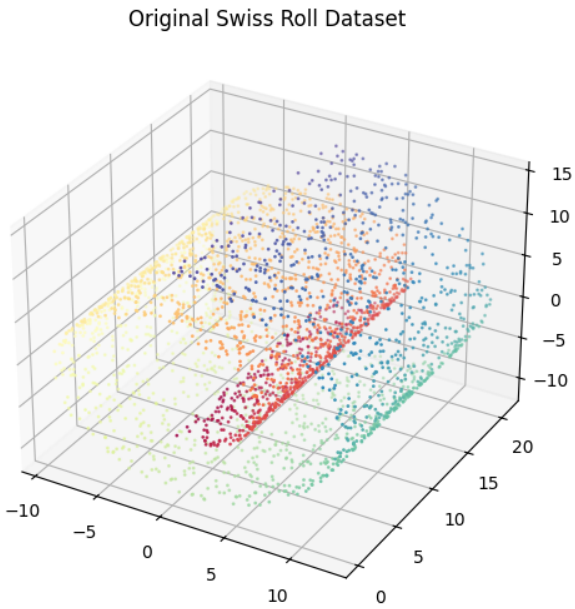


图 2: 原始数据集

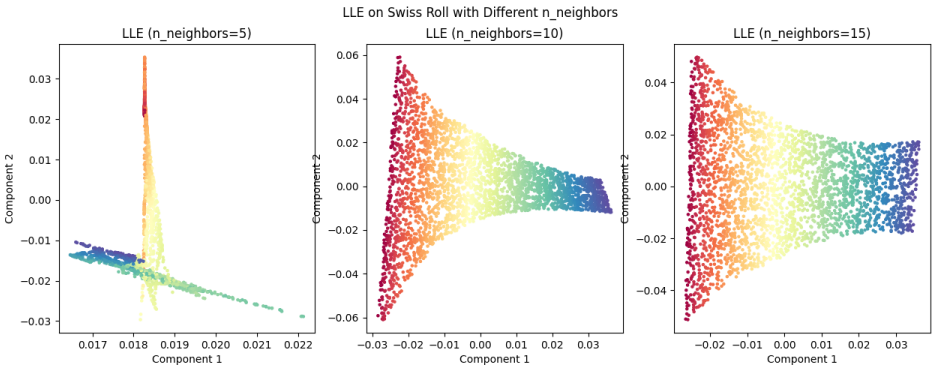


图 3: 多个不同参数下的可视化结果

- 最近邻数量 (n_neighbors) 的选择对降维效果有很大影响，过小可能导致局部结构信息丢失，过大则可能破坏局部线性假设
- 在大规模数据集上，计算最近邻和特征分解的开销较高
- 数据中的噪声可能导致局部线性假设失效，从而影响降维效果
- 如果嵌入空间维数选择不合理，可能无法充分表达数据的流形结构
- LLE 假设数据分布在低维流形上，若这一假设不成立，降维结果可能失真

二. (20 points) 特征选择

1. （12 points）使用 Wine 数据集，比较过滤式（基于互信息）和包裹式（RFE）特征选择方法的性能。要求：

(i) 实现两种特征选择方法，选择特征数量从 1 到全部特征

(ii) 使用交叉验证评估不同特征数量下的模型准确率

(iii) 绘制特征数量与准确率的关系图

(iv) 分析并比较：两种方法的最佳特征数量和对应准确率，计算并解释每个特征被选择的频率（特征重要性）

```python
def evaluate_model(X_selected, y):
    model = LogisticRegression(max_iter=200)
    scores = cross_val_score(model, X_selected, y, cv=5, scoring='accuracy')
    return np.mean(scores)
```

Listing 1: 评估对应选择特征的准确率

解:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import mutual_info_classif, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

data = load_wine()
X, y = data.data, data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

def evaluate_model(X_selected, y):
    model = LogisticRegression(max_iter=200)
    scores = cross_val_score(model, X_selected, y, cv=5, scoring='accuracy')
    return np.mean(scores)

mi_scores = mutual_info_classif(X_scaled, y)
mi_ranks = np.argsort(mi_scores)[::-1]

model = LogisticRegression(max_iter=200)
rfe = RFE(model, n_features_to_select=1)
rfe.fit(X_scaled, y)
rfe_ranks = np.argsort(rfe.ranking_)

mi_accuracies = []
rfe_accuracies = []

for k in range(1, X.shape[1] + 1):
    mi_selected_features = mi_ranks[:k]
    X_mi_selected = X_scaled[:, mi_selected_features]
```

```
    mi_acc = evaluate_model(X_mi_selected, y)
    mi_accuracies.append(mi_acc)

    rfe_selected_features = rfe_ranks[:k]
    X_rfe_selected = X_scaled[:, rfe_selected_features]
    rfe_acc = evaluate_model(X_rfe_selected, y)
    rfe_accuracies.append(rfe_acc)

plt.figure(figsize=(10, 6))
plt.plot(range(1, X.shape[1] + 1), mi_accuracies, label='Mutual Information', marker='o')
plt.plot(range(1, X.shape[1] + 1), rfe_accuracies, label='RFE', marker='s')
plt.xlabel('Number of Features')
plt.ylabel('Accuracy')
plt.title('Feature Selection Performance Comparison')
plt.legend()
plt.grid()
plt.show()

mi_best_k = np.argmax(mi_accuracies) + 1
mi_best_acc = mi_accuracies[mi_best_k - 1]
rfe_best_k = np.argmax(rfe_accuracies) + 1
rfe_best_acc = rfe_accuracies[rfe_best_k - 1]

print(f'互信息法最佳特征数量: {mi_best_k}, 对应准确率: {mi_best_acc:.4f}')
print(f'RFE法最佳特征数量: {rfe_best_k}, 对应准确率: {rfe_best_acc:.4f}')

mi_feature_importance = np.zeros(X.shape[1])
rfe_feature_importance = np.zeros(X.shape[1])

for k in range(1, X.shape[1] + 1):
    mi_selected_features = mi_ranks[:k]
    rfe_selected_features = rfe_ranks[:k]
    mi_feature_importance[mi_selected_features] += 1
    rfe_feature_importance[rfe_selected_features] += 1

print('互信息法特征选择频率:')
for i, freq in enumerate(mi_feature_importance):
    print(f'特征 {data.feature_names[i]}: {freq}')

print('\nRFE法特征选择频率:')
for i, freq in enumerate(rfe_feature_importance):
    print(f'特征 {data.feature_names[i]}: {freq}')
```

Listing 2: 两种特征选择方法实现

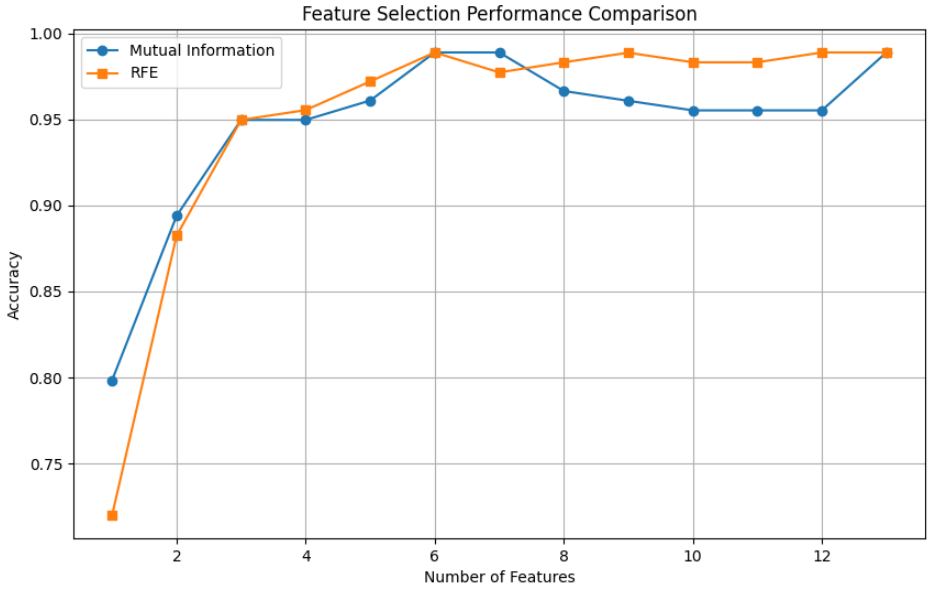


图 4: 特征数量与模型准确率的关系

```
互信息法最佳特征数量：6，对应准确率：0.9889
RFE法最佳特征数量：6，对应准确率：0.9889
```

图 5: 两种方法的最佳特征数量和对应准确率

```
互信息法特征选择频率：
特征 alcohol: 9.0
特征 malic_acid: 5.0
特征 ash: 1.0
特征 alcalinity_of_ash: 4.0
特征 magnesium: 3.0
特征 total_phenols: 7.0
特征 flavanoids: 13.0
特征 nonflavanoid_phenols: 2.0
特征 proanthocyanins: 6.0
特征 color_intensity: 11.0
特征 hue: 8.0
特征 od280/od315_of_diluted_wines: 10.0
特征 proline: 12.0
```

图 6: 互信息法特征选择频率

```
RFE法特征选择频率：
特征 alcohol: 10.0
特征 malic_acid: 4.0
特征 ash: 7.0
特征 alcalinity_of_ash: 6.0
特征 magnesium: 1.0
特征 total_phenols: 3.0
特征 flavanoids: 12.0
特征 nonflavanoid_phenols: 2.0
特征 proanthocyanins: 5.0
特征 color_intensity: 13.0
特征 hue: 9.0
特征 od280/od315_of_diluted_wines: 8.0
特征 proline: 11.0
```

图 7: RFE 法特征选择频率

2.（8 points）使用 L1 正则化的 Logistic 回归（LASSO）进行特征选择。

要求：

(i) 实现基于 LASSO 的特征选择，给出代码

(ii) 分析：

(a) 被选择的特征（系数非零）

(b) 特征的重要性排序（基于系数绝对值大小）

(c) 基于 Lasso 选择出特征（对应 Logistic 回归系数非 0），计算对应的模型准确率

(d) 对比相同特征数量下，三种特征选择方法的模型准确率

解:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import mutual_info_classif, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# 加载数据集
data = load_wine()
X, y = data.data, data.target

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 定义评估函数
def evaluate_model(X_selected, y):
    model = LogisticRegression(max_iter=200)
    scores = cross_val_score(model, X_selected, y, cv=5, scoring='accuracy')
    return np.mean(scores)

# 互信息特征选择
mi_scores = mutual_info_classif(X_scaled, y)
mi_ranks = np.argsort(mi_scores)[::-1]

# RFE特征选择
model = LogisticRegression(max_iter=200)
rfe = RFE(model, n_features_to_select=1)
rfe.fit(X_scaled, y)
rfe_ranks = np.argsort(rfe.ranking_)

# 初始化结果存储
mi_accuracies = []
rfe_accuracies = []

```

```python
# 评估不同特征数量下的模型准确率
for k in range(1, X.shape[1] + 1):
    # 互信息选择前k个特征
    mi_selected_features = mi_ranks[:k]
    X_mi_selected = X_scaled[:, mi_selected_features]
    mi_acc = evaluate_model(X_mi_selected, y)
    mi_accuracies.append(mi_acc)

    # RFE选择前k个特征
    rfe_selected_features = rfe_ranks[:k]
    X_rfe_selected = X_scaled[:, rfe_selected_features]
    rfe_acc = evaluate_model(X_rfe_selected, y)
    rfe_accuracies.append(rfe_acc)

# 使用 L1 正则化的 Logistic 回归 (LASSO)
lasso_model = LogisticRegression(penalty='l1', solver='saga', max_iter=500, C=1.0)
lasso_model.fit(X_scaled, y)

# 分析被选择的特征 (系数非零)
lasso_coefficients = lasso_model.coef_
lasso_selected_features = np.any(lasso_coefficients != 0, axis=0)
selected_feature_indices = np.where(lasso_selected_features)[0]
selected_feature_names = [data.feature_names[i] for i in selected_feature_indices]

# 特征的重要性排序 (基于系数绝对值大小)
lasso_importance = np.abs(lasso_coefficients).sum(axis=0)
sorted_indices = np.argsort(-lasso_importance)  # 按绝对值从大到小排序
sorted_feature_names = [data.feature_names[i] for i in sorted_indices]

# 基于 LASSO 选择出的特征，计算对应的模型准确率
X_lasso_selected = X_scaled[:, lasso_selected_features]
lasso_accuracy = evaluate_model(X_lasso_selected, y)

# 输出结果
print("LASSO 选择的特征:")
for i in selected_feature_indices:
    print(f"特征 {data.feature_names[i]}, 系数: {lasso_coefficients[:, i].sum():.4f}")

print("\n特征重要性排序:")
for i in sorted_indices:
    print(f"特征 {data.feature_names[i]}, 绝对值系数: {lasso_importance[i]:.4f}")

print(f"\n基于 LASSO 选择特征的模型准确率: {lasso_accuracy:.4f}")

# 绘制特征数量与准确率的关系图
plt.figure(figsize=(10, 6))
plt.plot(range(1, X.shape[1] + 1), mi_accuracies, label='Mutual Information', marker='o')
plt.plot(range(1, X.shape[1] + 1), rfe_accuracies, label='RFE', marker='s')
plt.axhline(lasso_accuracy, color='r', linestyle='--', label='LASSO')
plt.xlabel('Number of Features')
plt.ylabel('Accuracy')
plt.title('Feature Selection Performance Comparison')
```

```
plt.legend()
plt.grid()
plt.show()

# 分析最佳特征数量和对应准确率
mi_best_k = np.argmax(mi_accuracies) + 1
mi_best_acc = mi_accuracies[mi_best_k - 1]
rfe_best_k = np.argmax(rfe_accuracies) + 1
rfe_best_acc = rfe_accuracies[rfe_best_k - 1]
```

Listing 3: 基于 LASSO 的特征选择的实现

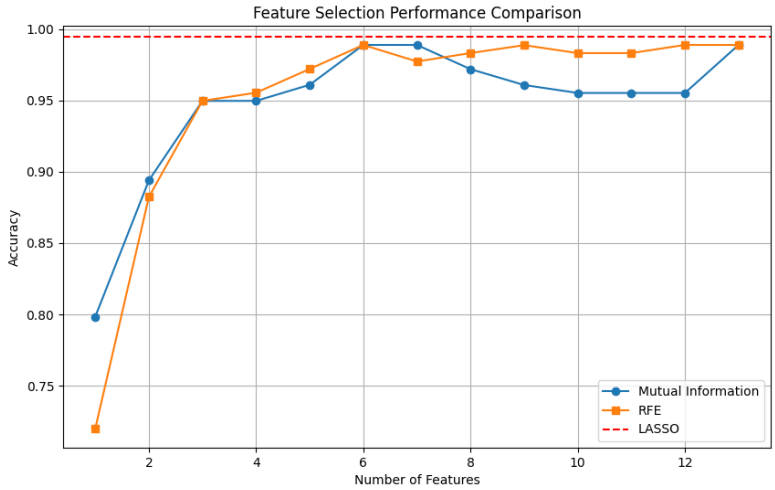


图 8: 特征数量与模型准确率的关系

```
基于 LASSO 选择特征的模型准确率：0.9944
互信息法最佳特征数量：6，对应准确率：0.9889
RFE法最佳特征数量：6，对应准确率：0.9889
```

图 9: 三种方法的最佳特征数量和对应准确率

```
LASSO 选择的特征：
特征 alcohol，系数：-1.6628
特征 malic_acid，系数：-0.3961
特征 ash，系数：-1.0993
特征 alcalinity_of_ash，系数：-0.9550
特征 flavanoids，系数：-2.2420
特征 nonflavanoid_phenols，系数：0.1782
特征 color_intensity，系数：-1.5881
特征 hue，系数：-0.5624
特征 od280/od315_of_diluted_wines，系数：0.0494
特征 proline，系数：0.1572
```

图 10: LASSO 选择的特征

```
特征重要性排序：
特征 proline，绝对值系数：2.4537
特征 flavanoids，绝对值系数：2.2420
特征 color_intensity，绝对值系数：1.7163
特征 alcohol，绝对值系数：1.6628
特征 od280/od315_of_diluted_wines，绝对值系数：1.5503
特征 hue，绝对值系数：1.3839
特征 ash，绝对值系数：1.0993
特征 alcalinity_of_ash，绝对值系数：1.0875
特征 malic_acid，绝对值系数：0.3961
特征 nonflavanoid_phenols，绝对值系数：0.1782
特征 magnesium，绝对值系数：0.0000
特征 total_phenols，绝对值系数：0.0000
特征 proanthocyanins，绝对值系数：0.0000
```

图 11: 特征重要性排序

三. (20 points) 半监督

1. 在本题中使用朴素贝叶斯模型和 SST2 数据集进行半监督 EM 算法的实践，代码前面部分如下，请补充完后续代码，只保留 10% 的标注数据，置信度设为 0.7，训练 5 轮，给出训练后模型在验证集上的分类结果 (10 points)

```python
import numpy as np
import random
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from datasets import load_from_disk
from scipy.sparse import vstack

random.seed(42)
np.random.seed(42)

data_path = "E:\\Codes\\sst2"
datasets = load_from_disk(data_path)
train_data = datasets['train']
valid_data = datasets['validation']

train_texts = [example['sentence'] for example in train_data]
train_labels = [example['label'] for example in train_data]
valid_texts = [example['sentence'] for example in valid_data]
valid_labels = [example['label'] for example in valid_data]

labeled_size = int(0.1 * len(train_texts))
indices = np.arange(len(train_texts))
np.random.shuffle(indices)

labeled_indices = indices[:labeled_size]
unlabeled_indices = indices[labeled_size:]

labeled_texts = [train_texts[i] for i in labeled_indices]
labeled_labels = [train_labels[i] for i in labeled_indices]
unlabeled_texts = [train_texts[i] for i in unlabeled_indices]

vectorizer = CountVectorizer()
X_labeled = vectorizer.fit_transform(labeled_texts)
X_unlabeled = vectorizer.transform(unlabeled_texts)
X_valid = vectorizer.transform(valid_texts)

model = MultinomialNB()
model.fit(X_labeled, labeled_labels)

confidence_threshold = 0.7

for epoch in range(5):
    print(f"Epoch {epoch + 1}...")

    probs = model.predict_proba(X_unlabeled)
```

```
    predicted_labels = model.predict(X_unlabeled)

    confident_indices = [i for i in range(len(probs)) if max(probs[i]) >= confidence_threshold]
    confident_texts = [unlabeled_texts[i] for i in confident_indices]
    confident_labels = [predicted_labels[i] for i in confident_indices]

    if len(confident_texts) == 0:
        print("No confident samples found. Stopping early.")
        break

    X_confident = vectorizer.transform(confident_texts)
    X_labeled = vstack([X_labeled, X_confident])
    labeled_labels.extend(confident_labels)

    unlabeled_indices = [i for i in range(len(unlabeled_texts)) if i not in confident_indices]
    unlabeled_texts = [unlabeled_texts[i] for i in unlabeled_indices]
    X_unlabeled = vectorizer.transform(unlabeled_texts)

    model.fit(X_labeled, labeled_labels)

valid_predictions = model.predict(X_valid)
accuracy = accuracy_score(valid_labels, valid_predictions)
print(f"Validation Accuracy: {accuracy:.4f}")

report = classification_report(valid_labels, valid_predictions, target_names=['negative', '
    positive'])
print("Classification Report:")
print(report)
```

Listing 4: 半监督 EM 算法的实践

```
Validation Accuracy: 0.7833
Classification Report:
              precision    recall  f1-score   support

    negative       0.80      0.74      0.77       428
    positive       0.77      0.82      0.79       444

    accuracy                           0.78       872
   macro avg       0.78      0.78      0.78       872
weighted avg       0.78      0.78      0.78       872
```

图 12: 训练后模型在验证集上的分类结果和性能

2. 伪标签的置信度大小对模型的训练结果会有一定的影响，通常会有固定置信度和动态设置置信度两种方式，请你完成这两种方式，并统计不同方式下每次迭代中伪标签的错误率，并分析这两种方式的优劣 (5 points)

```
import numpy as np
import random
```

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from datasets import load_from_disk
from scipy.sparse import vstack

random.seed(42)
np.random.seed(42)

data_path = "E:\\Codes\\sst2"
datasets = load_from_disk(data_path)
train_data = datasets['train']
valid_data = datasets['validation']

train_texts = [example['sentence'] for example in train_data]
train_labels = [example['label'] for example in train_data]
valid_texts = [example['sentence'] for example in valid_data]
valid_labels = [example['label'] for example in valid_data]

labeled_size = int(0.1 * len(train_texts))
indices = np.arange(len(train_texts))
np.random.shuffle(indices)

labeled_indices = indices[:labeled_size]
unlabeled_indices = indices[labeled_size:]

labeled_texts = [train_texts[i] for i in labeled_indices]
labeled_labels = [train_labels[i] for i in labeled_indices]
unlabeled_texts = [train_texts[i] for i in unlabeled_indices]

vectorizer = CountVectorizer()
X_labeled = vectorizer.fit_transform(labeled_texts)
X_unlabeled = vectorizer.transform(unlabeled_texts)
X_valid = vectorizer.transform(valid_texts)

model = MultinomialNB()
model.fit(X_labeled, labeled_labels)

dynamic_confidence_threshold = 0.7

dynamic_errors = []

for epoch in range(5):
    print(f"Epoch {epoch + 1}...")

    if X_unlabeled.shape[0] == 0:
        print("No unlabeled samples remaining. Stopping training.")
        break

    probs_dynamic = model.predict_proba(X_unlabeled)
    predicted_labels_dynamic = model.predict(X_unlabeled)

```

```
    confident_indices_dynamic = [i for i in range(len(probs_dynamic)) if max(probs_dynamic[i]) >= dynamic_confidence_threshold]
    confident_texts_dynamic = [unlabeled_texts[i] for i in confident_indices_dynamic]
    confident_labels_dynamic = [predicted_labels_dynamic[i] for i in confident_indices_dynamic]

    true_labels_dynamic = [train_labels[unlabeled_indices[i]] for i in confident_indices_dynamic]
    dynamic_error = sum(np.array(confident_labels_dynamic) != np.array(true_labels_dynamic)) / len(confident_labels_dynamic) if confident_labels_dynamic else 0
    dynamic_errors.append(dynamic_error)
    print(f"Dynamic confidence error rate: {dynamic_error:.4f}")

    X_confident_dynamic = vectorizer.transform(confident_texts_dynamic)
    X_labeled = vstack([X_labeled, X_confident_dynamic])
    labeled_labels.extend(confident_labels_dynamic)

    unlabeled_indices = [i for i in range(len(unlabeled_texts)) if i not in confident_indices_dynamic]
    unlabeled_texts = [unlabeled_texts[i] for i in unlabeled_indices]
    X_unlabeled = vectorizer.transform(unlabeled_texts)

    if dynamic_error > 0.5:
        dynamic_confidence_threshold = max(dynamic_confidence_threshold - 0.1, 0.5)
    elif dynamic_error < 0.3:
        dynamic_confidence_threshold = min(dynamic_confidence_threshold + 0.1, 1)

    model.fit(X_labeled, labeled_labels)

valid_predictions = model.predict(X_valid)
accuracy = accuracy_score(valid_labels, valid_predictions)
print(f"Validation Accuracy: {accuracy:.4f}")

print("Dynamic confidence error rates per epoch: [")
for error in dynamic_errors:
    print(f"    {error},")
print("]")
```

Listing 5: 半监督 EM 算法的动态置信度实践

```
Epoch 1...
Fixed confidence error rate: 0.1196
Epoch 2...
Fixed confidence error rate: 0.5073
Epoch 3...
Fixed confidence error rate: 0.5176
Epoch 4...
Fixed confidence error rate: 0.5196
Epoch 5...
Fixed confidence error rate: 0.5366
Validation Accuracy: 0.7833
Fixed confidence error rates per epoch: [
    0.11956450734893849,
    0.5073044602456367,
    0.5175824175824176,
    0.5195530726256983,
    0.5365853658536586,
]
```

图 13: 固定置信度每次迭代中伪标签的错误率

```
Epoch 1...
Dynamic confidence error rate: 0.1196
Epoch 2...
Dynamic confidence error rate: 0.5096
Epoch 3...
Dynamic confidence error rate: 0.4934
Epoch 4...
Dynamic confidence error rate: 0.4449
Epoch 5...
Dynamic confidence error rate: 0.4952
Validation Accuracy: 0.7833
Dynamic confidence error rates per epoch: [
    0.11956450734893849,
    0.5095961615353859,
    0.4934340118187787,
    0.4449438202247191,
    0.49523809523809526,
]
```

图 14: 动态置信度每次迭代中伪标签的错误率

- 固定置信度：
  - 优点：筛选规则简单稳定，不依赖错误率反馈，适合对训练过程较为清晰的任务，并且模型性能与验证集准确率表现稳定。
  - 缺点：对任务本身的适应性较差，可能导致伪标签质量在后期下降，在本题中第五轮错误率上升到 0.5366。
- 动态置信度：
  - 优点：自适应调整筛选规则，能够根据错误率动态调整筛选条件。在某些任务中可能更能保持伪标签样本质量，在本题中第四轮错误率 0.4449。
  - 缺点：实现复杂度较高，且错误率波动较大，对最终验证集性能的改进有限。

3. 修改代码，设置不同的迭代次数（如 3 次、5 次、15 次）。在验证集上分析：不同迭代次数下，模型性能如何变化？分析为什么在过多迭代的情况下，模型性能可能下降？(5 points)

```
Epoch 1...
Fixed confidence error rate: 0.0791
Epoch 2...
Fixed confidence error rate: 0.5049
Epoch 3...
Fixed confidence error rate: 0.5097
Validation Accuracy: 0.7810
```

图 15: 固定置信度每次迭代中伪标签的错误率

```
Epoch 1...
Fixed confidence error rate: 0.0791
Epoch 2...
Fixed confidence error rate: 0.5049
Epoch 3...
Fixed confidence error rate: 0.5097
Epoch 4...
Fixed confidence error rate: 0.5406
Epoch 5...
Fixed confidence error rate: 0.4968
Validation Accuracy: 0.7798
```

图 16: 动态置信度每次迭代中伪标签的错误率

```
Epoch 1...
Fixed confidence error rate: 0.0791
Epoch 2...
Fixed confidence error rate: 0.5049
Epoch 3...
Fixed confidence error rate: 0.5097
Epoch 4...
Fixed confidence error rate: 0.5406
Epoch 5...
Fixed confidence error rate: 0.4968
Epoch 6...
Fixed confidence error rate: 0.5000
Epoch 7...
Fixed confidence error rate: 0.4643
Epoch 8...
Fixed confidence error rate: 0.7778
Epoch 9...
Fixed confidence error rate: 0.6250
Epoch 10...
Fixed confidence error rate: 0.5000
Epoch 11...
Fixed confidence error rate: 1.0000
Epoch 12...
No confident samples found. Stopping early.
Validation Accuracy: 0.7787
```

图 17: 训练后模型在验证集上的分类结果和性能

随着迭代次数增加，错误率在第 8-12 轮逐渐上升，甚至在第 11 轮达到 1.000。原因分析：

- 初期伪标签样本质量较高，因为模型对置信度高的样本有较高的预测准确率。随着迭代次数增加，未标注数据中剩余样本的分布更接近模型难以区分的样本区域（置信度低且容易出错），伪标签的错误率逐步上升。
- 每轮迭代中，错误伪标签样本可能被模型错误学习，并影响后续模型的预测。错误样本的累积最终导致模型性能下降。
- 在后期，未标注样本逐渐被筛选完毕，剩余样本数量和多样性不足，导致模型无法进一步学习有效信息。
- 过多迭代可能导致模型过拟合于伪标签样本中的噪声，而不是学习到有用的模式。

四. (20 points) 概率图模型

1. 证明 $a \perp\!\!\!\perp (b,c) \mid d$ 蕴含 $a \perp\!\!\!\perp b \mid d$。(5 points)

解:

$a \perp\!\!\!\perp (b,c) \mid d$ 表示在给定 d 的条件下，a 与 (b,c) 联合分布独立，即 $P(a,b,c \mid d) = P(a \mid d)P(b,c \mid d)$

要证明 $a \perp\!\!\!\perp b \mid d$，需要证明：$P(a,b \mid d) = P(a \mid d)P(b \mid d)$

从联合概率的边际化：$P(a,b \mid d) = \sum_c P(a,b,c \mid d)$

代入得：$P(a,b \mid d) = \sum_c P(a \mid d)P(b,c \mid d) = P(a \mid d)\sum_c P(b,c \mid d) = P(a \mid d)P(b \mid d)$

综上所述，得证

2. 假设你有一组 d 个二元随机变量 $\{X_1, ..., X_d\}$。(5 points)

(i) 在不做任何独立性假设的情况下，完全描述联合分布所需的最小参数个数是多少?
提示: 由于总概率之和必须为 1，所需的参数个数比结果的总数少一个。

解:

对于每个二元随机变量 X_i，它有两种可能的取值

考虑所有变量的联合分布：每个变量有 2 种可能的取值，总共有 d 个变量，因此所有可能的组合数是 2^d

对于这些组合，每个都对应一个概率值。由于概率之和必须为 1：$\sum P(\text{所有组合}) = 1$，这意味着知道了 $2^d - 1$ 个概率值，最后一个就能通过 1 减去其他概率之和得到所以我们只需要指定 $2^d - 1$ 个参数

因此，完全描述 d 个二元随机变量的联合分布所需的最小参数个数是 $2^d - 1$。

(ii) 假设这些随机变量的结构为马尔可夫链，其中每个 X_i 仅依赖于 X_{i-1}。在这种情况下，所需的最小参数个数是多少?

解:

要完整描述这个马尔可夫链，我们需要：

(1) 初始分布 $P(X_1)$：需要 1 个参数。因为 X_1 是二元的，$P(X_1 = 1)$ 确定后 $P(X_1 = 0) = 1 - P(X_1 = 1)$

(2) 转移概率 $P(X_i|X_{i-1})$：对每个 i 从 2 到 d，对于每个 X_{i-1} 的取值 (0 或 1)，需要 1 个参数来描述 X_i 的条件分布，因此每个转移矩阵需要 2 个参数。总共需要 $2(d-1)$ 个参数描述所有转移概率

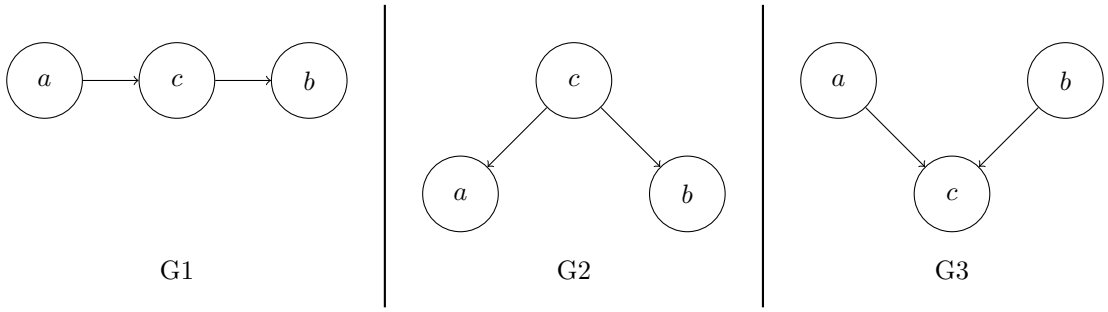
总参数个数：$2d - 1$

(iii) 从不做独立性假设到引入马尔可夫假设，参数复杂度是如何变化的?

解:

不做独立性假设时：参数个数为 $2^d - 1$，指数复杂度 $O(2^d)$；马尔可夫假设下：参数个数为 $2d - 1$，线性复杂度 $O(d)$。从指数复杂度降到了线性复杂度，这是非常显著的降低

3. 考虑以下三种结构各异的图模型。



请将每个情境与最合适的图模型进行匹配。(5 points)

(i) 一个家庭的旅行决定 (c) 会受到父母的工作安排 (a) 孩子的学校假期 (b) 的影响。

解: G3

(ii) 破纪录的大雪 (c) 会同时刺激滑雪度假村的预订量 (a) 和冬季服装的需求 (b)。

解: G2

(iii) 个人的锻炼习惯 (a) 会影响自身的能量水平 (c)，进而影响工作效率 (b)。

解: G1

(iv) 一个地区的气候 (a) 决定了生长的植被类型 (c)，而植被类型又会影响野生动物的数量 (b)。

解: G1

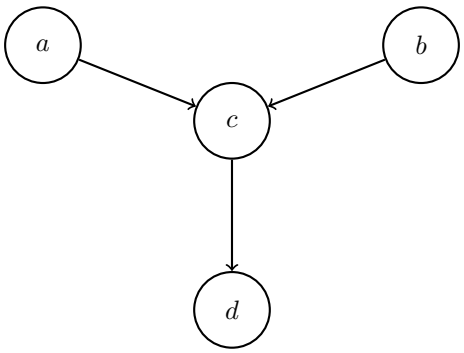
(v) 一个国家的经济稳定性 (c) 会影响就业率 (a) 和消费者的消费习惯 (b)。

解: G2

(vi) 餐厅的受欢迎程度 (c) 取决于食品质量 (a) 和社交媒体的曝光度 (b)。

解: G3

4. 考虑以下有向图，其中所有变量均为未观测变量。(5 points)



(i) 在给定的图中，如何将联合分布 $p(a, b, c, d)$ 表示为边际分布和条件分布的组合?

解: $p(a,b,c,d) = p(a) \cdot p(b) \cdot p(c \mid a,b) \cdot p(d \mid c)$

(ii) 假设 a, b, c, d 是二元随机变量，求建模该联合分布所需的最小参数个数。

解:

$p(a)$: 二元变量需要 1 个参数
$p(b)$: 二元变量需要 1 个参数
$p(c|a,b)$: 需要 4 个参数，因为对 a，b 的每种组合（00,01,10,11）都需要一个条件概率
$p(d|c)$: 需要 2 个参数，因为 c 的每个取值都需要一个条件概率

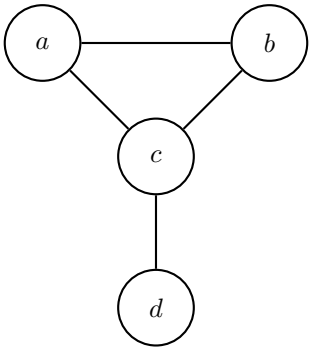
总计：$1+1+4+2=8$ 个参数

(iii) 证明 a 和 b 是相互独立的，即 $a \perp\!\!\!\perp b$。

解: 在图中，a 和 b 之间唯一可能的路径是通过 c，但 abc 是一个 V 型结构，当 c 未观测时，a 和 b 相互独立

(iv) 假设对于图中任意节点 x , 都有 $p(x \mid \mathrm{pa}(x)) \neq p(x)$, 其中 $\mathrm{pa}(\cdot)$ 表示节点 x 的父节点集合。证明当观测到 d 时, a 和 b 不再相互独立, 即 $a \not\perp\!\!\!\perp b \mid d$

解: 道德图：



如果观测到 d，即条件化 d，a 和 b 在道德图中有边连接，并未被 d 分入两个连通分支，因此 a 和 b 不相互独立, 即 $a \not\perp\!\!\!\perp b \mid d$

五. (20 points) 强化学习

在本问题中，你将思考如何通过在马尔可夫决策过程（MDP）中连续做决策来最大化奖励，并深入了解贝尔曼方程——解决和理解 MDP 的核心方程。

考虑经典的网格世界 MDP，其中智能体从单元格 (1, 1) 开始，并在环境中导航。在这个世界中，智能体每个格子里可以采取四个动作：上、下、左、右。格子用 (水平，垂直) 来索引；也就是说，单元格 (4, 1) 位于右下角。世界的转移概率如下：如果智能体采取一个动作，它将以 0.8 的概率移动到动作的方向所在的格子，并以 0.1 的概率滑到动作的相对右或左的方向。如果动作（或滑动方向）指向一个没有可通过的格子（即边界或 (2, 2) 格子的墙壁），那么该动作将保持智能体处于当前格子。例如，如果智能体在 (3, 1) 位置，并采取向上的动作，它将以 0.8 的概率移动到 (3, 2)，以 0.1 的概率移动到 (2, 1)，以 0.1 的概率移动到 (4, 1)。如果智能体在 (1, 3) 位置并采取右移动作，它将以 0.8 的概率移动到 (2, 3)，以 0.1 的概率移动到 (1, 2)，以 0.1 的概率停留在 (1, 3)。当智能体到达定义的奖励状态时（在 (4, 2) 和 (4, 3) 单元格），智能体将获得相应的奖励，并且本次回合结束。

回顾计算 MDP 中每个状态的最优价值, $V^*(s)$ 的贝尔曼方程，其中我们有一组动作 A，一组状态 S，每个状态的奖励值 $R(s)$，我们的世界的转移动态 $P(s'|s,a)$，以及折扣因子 γ：

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

最后，我们将策略表示为 $\pi(s) = a$ 其中策略 π 指定了在给定状态下采取的行动。

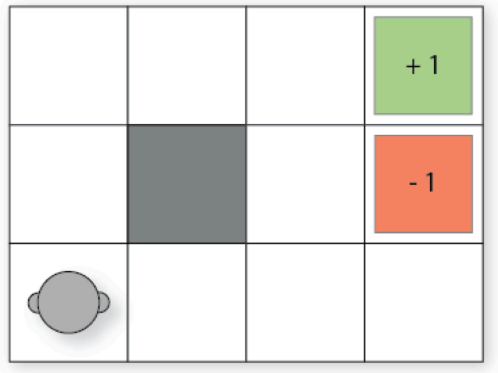


图 18: 经典网格世界

(a) 考虑一个智能体从单元格 (1, 1) 开始，在第 1 步和第 2 步分别采取向上和向上的动作。计算在每个时间步内，根据这一动作序列，智能体可以到达哪些单元格，以及到达这些单元格的概率。(6 points)

解:

第 1 步：$P((1,2)) = 0.8$，$P((2,1)) = 0.1$，$P((1,1)) = 0.1$

第 2 步：

如果在（1，2）：$P((1,3)) = 0.64$，$P((1,2)) = 0.16$

如果在（2，1）：$P((2,1)) = 0.08$，$P((1,1)) = 0.01$，$P((3,1)) = 0.01$

如果在（1，1）：$P((1,2)) = 0.08$，$P((2,1)) = 0.01$，$P((1,1)) = 0.01$

$P((1,3)) = 0.64$，$P((1,2)) = 0.24$，$P((2,1)) = 0.09$，$P((1,1)) = 0.02$，$P((3,1)) = 0.01$

(b) 考虑当前没有奖励值的所有状态的奖励函数 $R(s)$（即除了 (4, 2) 和 (4, 3) 以外的每个单元格）。定义在以下奖励值下，智能体的最优策略：(i.) $R(s) = 0$, (ii.) $R(s) = -2.0$, and (iii.) $R(s) = 1.0$. 你可以假设折扣因子接近 1，例如 0.9999。画出网格世界并标出在每个状态下应采取的动作可能会对你有所帮助（记住，策略是在 MDP 中对所有状态进行定义的!）（7 points）

注意: 你不需要算法上计算最优策略。你必须列出每种情况的完整策略，但只需要提供直观的理由。

解:

（1） $R(s) = 0$：在非终止状态中，每走一步不产生正负奖励，只有当到达（4，3）时获得正奖励（+1），到达（4，2）时获得负奖励（-1）。在这种情况下，智能体将尝试走一条安全且稳定到达（4，3）的路线。由于（2，2）是墙壁，最可能的选择是从（1，1）开始，向上走到（1，2），再到（1，3），然后往右走经过（2，3），（3，3），最终抵达 (4,3)。整体策略是朝着（4，3）方向移动，同时避免靠近（4，2）防止因随机滑动而掉入负终止状态。

（2） $R(s) = -2.0$：每一个非终止状态都有一个-2 的惩罚，相当于每走一步都会使总回报减少。这种设定会使智能体非常厌恶长时间拖延，因为每多走一步就要多承受-2 的成本。在这种情况下，即使到达（4，3）最终也会获得-9，如果在路上需要好几步才能绕过墙壁到达目标，那么中间的累计惩罚会变得很大。与之相比，如果直接走到（4，2）获得-1 终止奖励，但可能只需要 4 步左右，总成本为 4 * (-2) + (-1) = -9，和到（4，3）差不多。因此整体策略是快速走向最近的终止状态。

（3） R(s)=1.0：每经过一个非终止状态都能获得 +1 的奖励。这意味着只要不断地走下去，不结束回合，就可以累积大量正奖励。折扣因子接近 1 时（例如 0.9999），未来奖励几乎不被减小，意味着无限期地在网格内移动累积奖励将产生巨大价值。因此整体策略会变成在所有状态中采取的行动都以“远离终止状态并保持存活”为目标，多在安全的非终止区内迂回移动，从而无限接近无限累计的正奖励。

(c) 有时，MDP 的奖励函数形式为 $R(s, a)$ 它依赖于所采取的动作，或者奖励函数形式为 $R(s, a, s')$ ，它还依赖于结果状态。写出这两种形式的最优价值函数的贝尔曼方程。（7 points）

解:

当奖励函数为 $R(s, a)$ 时：

$$V^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right].$$

当奖励函数为 $R(s, a, s')$ 时：

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a) \left[R(s, a, s') + \gamma V^*(s') \right].$$