**实验设计：基于ML模型的数据清洗算法对下游聚类应用效果的影响（简要思路） 10/04/2024**

**1. 实验目标**

实验旨在评估三种基于ML模型的数据清洗算法（Raha-Baran、Holoclean、MLNClean）对六种聚类算法（K-Means、GMM、AP、HC、OPTICS、BIRCH）的影响，并测试不同错误率（10%、30%、50%、70%、90%）下的聚类表现。

**2. 实验数据**

**2.1 数据集（实验所需的脏数据集和干净数据集已经准备好，但是还需要进行不同百分比的错误注入）**

使用干净数据集（ground truth clean data）作为基础，并生成不同错误率的脏数据集。

**2.2 错误注入 （这里总结了论文中的常见错误，并做了python代码示例，出于简要考虑，未必全部应用所有错误类型）**

**2.2.1语法错误 (Syntactic Errors)**

**① 缺失值 (Missing Values)**

随机移除部分数据。

def inject\_missing\_values(df, error\_rate):

df\_copy = df.copy()

n\_missing = int(error\_rate \* df\_copy.size)

missing\_indices = (np.random.randint(0, df\_copy.shape[0], n\_missing),

np.random.randint(0, df\_copy.shape[1], n\_missing))

df\_copy.values[missing\_indices] = np.nan

return df\_copy

**② 拼写错误 (Typographical Errors)**

引入拼写错误，打乱字符串中的字符顺序。

def inject\_typo\_errors(df, error\_rate, columns):

df\_copy = df.copy()

for col in columns:

n\_typos = int(error\_rate \* df\_copy[col].size)

typo\_indices = np.random.choice(df\_copy.index, n\_typos, replace=False)

for idx in typo\_indices:

val = df\_copy.at[idx, col]

if isinstance(val, str):

typo\_val = ''.join(random.sample(val, len(val))) # 打乱字符串顺序

df\_copy.at[idx, col] = typo\_val

return df\_copy

**③ 格式错误 (Formatting Errors)**

引入不一致的格式，例如日期或数值格式。

def inject\_format\_errors(df, columns, formats):

df\_copy = df.copy()

for col in columns:

for idx in df\_copy.index:

if isinstance(df\_copy.at[idx, col], str):

df\_copy.at[idx, col] = random.choice(formats) # 随机选择一种格式应用

return df\_copy

**2.2.2 语义错误 (Semantic Errors)**

**① 错误的上下文关系 (Contextual Errors)**

打乱属性之间的上下文关系，例如将城市与不匹配的国家对应。

def inject\_contextual\_errors(df, error\_rate, col1, col2):

df\_copy = df.copy()

n\_errors = int(error\_rate \* df\_copy.shape[0])

error\_indices = np.random.choice(df\_copy.index, n\_errors, replace=False)

for idx in error\_indices:

df\_copy.at[idx, col2] = random.choice(df\_copy[col2].unique()) # 打乱与col1的逻辑关系

return df\_copy

**② 违反属性依赖 (Violated Attribute Dependencies)**

破坏属性间的逻辑关系。

def inject\_vad\_errors(df, error\_rate, dependencies):

df\_copy = df.copy()

for dep in dependencies:

col1, col2 = dep

n\_violations = int(error\_rate \* df\_copy.shape[0])

violation\_indices = np.random.choice(df\_copy.index, n\_violations, replace=False)

for idx in violation\_indices:

df\_copy.at[idx, col2] = random.choice(df\_copy[col2].unique()) # 破坏 col1 和 col2 的依赖关系

return df\_copy

**2.2.3 模式违规 (Pattern Violations)**

**不符合特定模式的数据 (Pattern Violation Errors)**

引入不符合特定模式的错误，例如日期格式。

def inject\_pattern\_errors(df, columns, pattern\_func):

df\_copy = df.copy()

for col in columns:

for idx in df\_copy.index:

if isinstance(df\_copy.at[idx, col], str):

df\_copy.at[idx, col] = pattern\_func(df\_copy.at[idx, col]) # 应用不符合模式的格式

return df\_copy

**2.2.4规则违规 (Rule Violations)**

**违反业务规则 (Business Rule Violations)**

打破数值或业务逻辑的规则，如价格不能为负数。

def inject\_rule\_violations(df, error\_rate, column, rule\_func):

df\_copy = df.copy()

n\_violations = int(error\_rate \* df\_copy.shape[0])

violation\_indices = np.random.choice(df\_copy.index, n\_violations, replace=False)

for idx in violation\_indices:

df\_copy.at[idx, column] = rule\_func(df\_copy.at[idx, column]) # 违反业务规则

return df\_copy

**2.2.5 基于知识库的违规 (Knowledge Base Violations)**

**知识库冲突 (Knowledge Base Violations)**

引入与外部知识库冲突的数据错误。

def inject\_knowledge\_base\_violations(df, error\_rate, column, knowledge\_base):

df\_copy = df.copy()

n\_violations = int(error\_rate \* df\_copy.shape[0])

violation\_indices = np.random.choice(df\_copy.index, n\_violations, replace=False)

for idx in violation\_indices:

df\_copy.at[idx, column] = random.choice(knowledge\_base) # 与外部知识库不匹配

return df\_copy

**错误类型与代码示例总结**

* **语法错误**：缺失值、拼写错误、格式错误。
* **语义错误**：上下文关系错误、属性依赖破坏。
* **模式违规**：不符合特定模式的数据。
* **规则违规**：违反业务规则（如价格不能为负数）。
* **基于知识库的违规**：与外部知识库不匹配的错误。

**2.3 错误率**

根据10%、30%、50%、70%、90%的错误率生成不同的脏数据集。

**3. 数据清洗方法**

**3.1 数据清洗算法**

本实验将测试三种数据清洗算法：

1. **Raha-Baran**：数据驱动的清洗方法。
2. **Holoclean**：基于概率推断的清洗算法。
3. **MLNClean**：基于马尔可夫逻辑网络的清洗算法。

**清洗数据的示例代码：**

# Raha-Baran 清洗

from raha\_baran\_cleaning\_lib import RahaBaranCleaner

cleaner\_raha = RahaBaranCleaner()

df\_cleaned\_raha\_10 = cleaner\_raha.clean(df\_typo\_10)

# Holoclean 清洗

from holoclean\_lib import HoloClean

holoclean = HoloClean()

df\_cleaned\_holo\_10 = holoclean.clean(df\_typo\_10)

# MLNClean 清洗

from mlnclean\_lib import MLNClean

mlnclean = MLNClean()

df\_cleaned\_mln\_10 = mlnclean.clean(df\_typo\_10)

**4. 聚类算法**

**4.1 聚类算法选择**

我们将使用六种常见的聚类算法对清洗后的数据进行测试：

1. **K-Means**：基于距离的划分聚类算法。
2. **GMM (Gaussian Mixture Model)**：基于高斯分布的聚类算法。
3. **Affinity Propagation (AP)**：基于消息传递的聚类方法。
4. **Hierarchical Clustering (HC)**：层次聚类，生成聚类树。
5. **OPTICS**：基于密度的聚类算法，适用于处理噪声和不规则簇。
6. **BIRCH**：基于层次结构的聚类算法，适用于大规模数据集。

**聚类算法代码示例：**

from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation, OPTICS, Birch

from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette\_score

# K-Means 聚类

kmeans = KMeans(n\_clusters=5, random\_state=42)

labels\_kmeans = kmeans.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_kmeans = silhouette\_score(df\_cleaned\_raha\_10, labels\_kmeans)

print(f"K-Means 轮廓系数: {silhouette\_kmeans}")

# GMM 聚类

gmm = GaussianMixture(n\_components=5, random\_state=42)

labels\_gmm = gmm.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_gmm = silhouette\_score(df\_cleaned\_raha\_10, labels\_gmm)

print(f"GMM 轮廓系数: {silhouette\_gmm}")

# Affinity Propagation 聚类

ap = AffinityPropagation()

labels\_ap = ap.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_ap = silhouette\_score(df\_cleaned\_raha\_10, labels\_ap)

print(f"Affinity Propagation 轮廓系数: {silhouette\_ap}")

# 层次聚类

hc = AgglomerativeClustering(n\_clusters=5)

labels\_hc = hc.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_hc = silhouette\_score(df\_cleaned\_raha\_10, labels\_hc)

print(f"层次聚类 轮廓系数: {silhouette\_hc}")

# OPTICS 聚类

optics = OPTICS()

labels\_optics = optics.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_optics = silhouette\_score(df\_cleaned\_raha\_10, labels\_optics)

print(f"OPTICS 轮廓系数: {silhouette\_optics}")

# BIRCH 聚类

birch = Birch(n\_clusters=5)

labels\_birch = birch.fit\_predict(df\_cleaned\_raha\_10)

silhouette\_birch = silhouette\_score(df\_cleaned\_raha\_10, labels\_birch)

print(f"BIRCH 轮廓系数: {silhouette\_birch}")

**5. 实验流程**

**5.1 数据生成与清洗**

1. 通过注入不同类型的错误生成脏数据集，分别设置10%、30%、50%、70%、90%的错误率。
2. 使用三种数据清洗算法（Raha-Baran、Holoclean、MLNClean）对脏数据进行清洗，生成清洗后的数据集。（此处可能需要用户标签和配置，例如标记采样的正确性，完整性约束，值约束等，需要根据具体的算法执行步骤确定。某些算法有预训练步骤，初步试验可以忽略，直接按照论文作者在github上的源码要求进行实验）

**5.2 聚类任务**

1. 针对清洗前和清洗后的数据，使用六种聚类算法（K-Means、GMM、AP、HC、OPTICS、BIRCH）进行聚类。
2. 使用轮廓系数（Silhouette Coefficient）、调整兰德指数（ARI）、簇内均方误差（WCSS）等指标评估聚类效果。

**5.3 实验评价指标（有待补充，这三个只是评价聚类算法的通用参数，可以参考更多文献和算法作出补充）**

* **轮廓系数（Silhouette Coefficient）**：衡量聚类的紧密度和分离度。
* **调整兰德指数（Adjusted Rand Index, ARI）**：在有真实标签的情况下，评估聚类结果与真实分类的相似度。
* **簇内均方误差（Within-cluster Sum of Squares, WCSS）**：衡量聚类簇内部的紧密性。