数据挖掘互评作业 4：离群点分析与异常检测

姓名：曹健　　学号：3120190978

一．问题

本次作业将从 Anomaly Detection Meta-Analysis Benchmarks 提供的 benchmark 数据集中任选两个进行分析。

可以使用 Python Outlier Detection (PyOD)或其他已知的工具包来完成分析工作。

二．数据

1. 数据集选择：wine_benchmarks 数据集、abalone_benchmarks 数据集
2. 数据集描述：

   两个数据集都分为两部分数据，分别为 meta_data 文件夹中的原始整体数据以及 benchmarks 文件夹中进行划分后的多个子数据集，子数据集之间相互独立。

三、数据集预处理（以数据集 wine_benchmarks 举例）

1. 找出公共列

观察发现每个数据集中的各 csv 文件大都具有不同的列，为了更有效地进行数据挖掘，我们提取所有文件中共有的列名。

```python
# 提取两个列表的共同元素
def extra_element(list1, list2):
    set1 = set(list1)
    set2 = set(list2)
    same_set = set1.intersection(set2)
    return list(same_set)


# 提取文件
def load_file(file_path):
    file_list = []
    i = 0
    for _ in os.listdir(file_path):
        file_list.append(file_path + '/' + _)
    return file_list


# 提取各文件公共列
def extra_column(file_list):
    list1 = pd.read_csv(file_list[0], index_col=0).columns
    print("提取所有csv文件中的公共列：")
    for file in tqdm(file_list[1:]):
        list2 = pd.read_csv(file, index_col=0).columns
        list1 = extra_element(list1, list2)
    print("公共列为：", list1)
    return list1


if __name__ == "__main__":
```

提取结果即公共列为：

```
# columns = extra_column(file_list)
# 为节约运行时间，直接将上面得到的公共列付给columns变量
columns = ['density', 'fixed.acidity', 'original.label', 'free.sulfur.dioxide', 'volatile.acidity', 'chlorides',
       'total.sulfur.dioxide', 'pH', 'alcohol', 'sulphates', 'citric.acid', 'diff.score', 'residual.sugar']
```

2. 数据划分

将数据集按照 7:3 的比例划分为训练集与测试集，在训练集上对各种检测离群点的方法进行训练，再在测试集上查看各种方法的效果，并进行对比。

3. 算法选择

实验中通过 PyOD 工具，使用 CBLOF、KNN、PCA、IForest 四种算法来检测异常，并计算每种算法在各个 csv 文件中的 ROC 值和 PRN 值。

CBLOF 算法

CBLOF 算法是基于聚类和邻近性的异常检测方法，它将数据分为香型集群和大型集群，然后根据点所属的簇的大小以及到最近的大簇的距离来计算异常分数。

KNN 算法

KNN 算法是典型的基于邻近性的异常检测算法，对于任何数据点，到第 k 个最近邻居的距离可以被视为远离分数，PyOD 支持三个 KNN 探测器，分别为最大 KNN、均值 KNN 以及中位数 KNN，区别在于作为离群值得分的距离的计算方法。在这里我们用默认的最大 KNN 算法，也就是使用最大距离作为离群值。

PCA 算法

PCA 算法是一种最常见的数据降维的方法，它可以将原数据进行线性变换，并找出数据中信息含量最大的主要成分，去除信息含量较低的成分，从而减少冗余，降低噪音。通常在异常检测的语境里，噪音（noise）、离群点（outlier）和 异常值（anomaly）是同一件事情的不同表述。所以，PCA 既然可以识别噪音，自然也可以检测异常。

Isolation Forest 算法

该方法是基于聚类的方法，使用一组树完成数据分区。隔离森林提供了一个异常分数用来查看结构中点的隔离程度。

四、训练与测试（以数据集 wine_benchmarks 举例）

1. 对每个 csv 文件，分别将其换分为训练集和测试集。

```
# 划分数据集
def split_data(file_list, columns):
    for file in file_list:
        print(file)
        data = pd.read_csv(file, index_col=0)
        #data = data[data['ground.truth'] == 'nominal']
        #data = data[columns]
        data_len = len(data)
        data.loc[data['ground.truth'] == 'anomaly','ground.truth'] = 1
        data.loc[data['ground.truth'] == 'nominal','ground.truth'] = 0

        x = data[columns]
        y = data['ground.truth']
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

    return x_train, x_test, y_train, y_test
```

2. 在训练集和测试集上分别计算 ROC 值和 PRN 值。

```
def calculate(method, total_roc, total_prn, x_train, x_test, y_train, y_test):
    if method == 'KNN':
        clf = KNN()
    elif method == 'CBLOF':
        clf = CBLOF()
    elif method == 'PCA':
        clf = PCA()
    else:
        clf = IForest()
    clf.fit(x_train)  # 使用x_train训练检测器clf

    # 返回训练数据x_train上的异常标签和异常分值
    y_train_pred = clf.labels_  # 返回训练数据上的分类标签 (0: 正常值, 1: 异常值)
    y_train_scores = clf.decision_scores_  # 返回训练数据上的异常值 (分值越大越异常)
    print("On train Data:")
    evaluate_print(method, y_train, y_train_scores)

    # 用训练好的clf来预测未知数据中的异常值
    y_test_pred = clf.predict(x_test)  # 返回未知数据上的分类标签 (0: 正常值, 1: 异常值)
    y_test_scores = clf.decision_function(x_test)  # 返回未知数据上的异常值 (分值越大越异常)
    print("On Test Data:")
    evaluate_print(method, y_test, y_test_scores)

    y_true = column_or_1d(y_test)
    y_pred = column_or_1d(y_test_scores)
    check_consistent_length(y_true, y_pred)

    roc = np.round(roc_auc_score(y_true, y_pred), decimals=4),
    prn = np.round(precision_n_scores(y_true, y_pred), decimals=4)

    total_roc.append(roc)
    total_prn.append(prn)
```

3. 结果可视化。

```
# 结果可视化
def visualisation(roc_all, prn_all, names):
    plt.figure(figsize=(10, 10), dpi=80)
    # 柱子总数
    N = 4
    # 包含每个柱子对应值的序列
    values1 = roc_all
    values2 = prn_all
    # 包含每个柱子下标的序列
    index = np.arange(N)
    # 绘制柱状图
    p1 = plt.bar(index, values1, label="ROC")
    p2 = plt.bar(index, values2, label="PRN")
    plt.xlabel('algorithm')
    plt.ylabel('ROC')
    plt.title('')
    plt.xticks(index, ('KNN', 'CBLOF', 'PCA', 'IForest'))
    plt.legend(loc="upper right")
    plt.show()
```

五、实验结果分析

1. wine_benchmarks 数据集

该数据集共包括 1210 个.csv 数据文件（注：文件的最大标号为 1680，但实际只包括

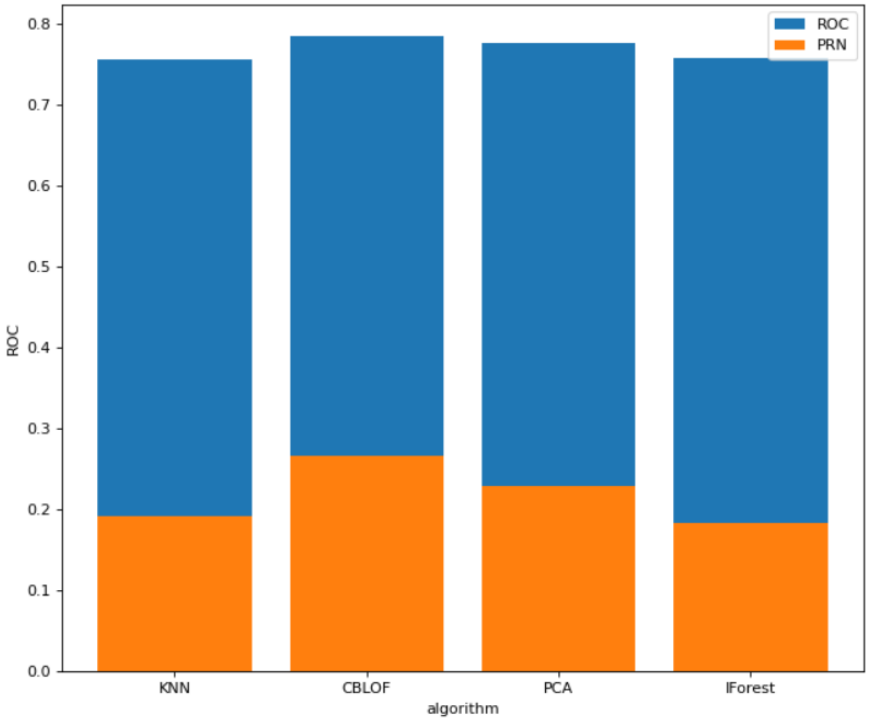
了 1210 个文件），其中各个数据文件中共有的公共列有 13 个，分别为：

columns = ['density', 'fixed.acidity', 'original.label', 'free.sulfur.dioxide', 'volatile.acidity', 'chlorides', 'total.sulfur.dioxide', 'pH', 'alcohol', 'sulphates', 'citric.acid', 'diff.score', 'residual.sugar']

训练数据集、测试数据集在不同算法下的 ROC 值和 PRN 值如下所示：

```
On train Data:
knn ROC:0.7526, precision @ rank n:0.2411
On Test Data:
knn ROC:0.7566, precision @ rank n:0.1927
On train Data:
cblof ROC:0.777, precision @ rank n:0.2796
On Test Data:
cblof ROC:0.7851, precision @ rank n:0.2661
On train Data:
pca ROC:0.7673, precision @ rank n:0.2553
On Test Data:
pca ROC:0.7773, precision @ rank n:0.2294
On train Data:
iforest ROC:0.7523, precision @ rank n:0.2128
On Test Data:
iforest ROC:0.7577, precision @ rank n:0.1835
```

我们将各算法在测试集上的实验结果进行可视化对比，得到如下对比图：



从图中我们可以发现，在 ROC 值方面，CBLOF 算法的实验结果最佳；在 PRN 值上，同样也是 CBLOF 算法的实验结果最佳。也可以大致看出，在两个评测值上，四种算法的优劣状态基本一致。

2. abalone_benchmarks 数据集

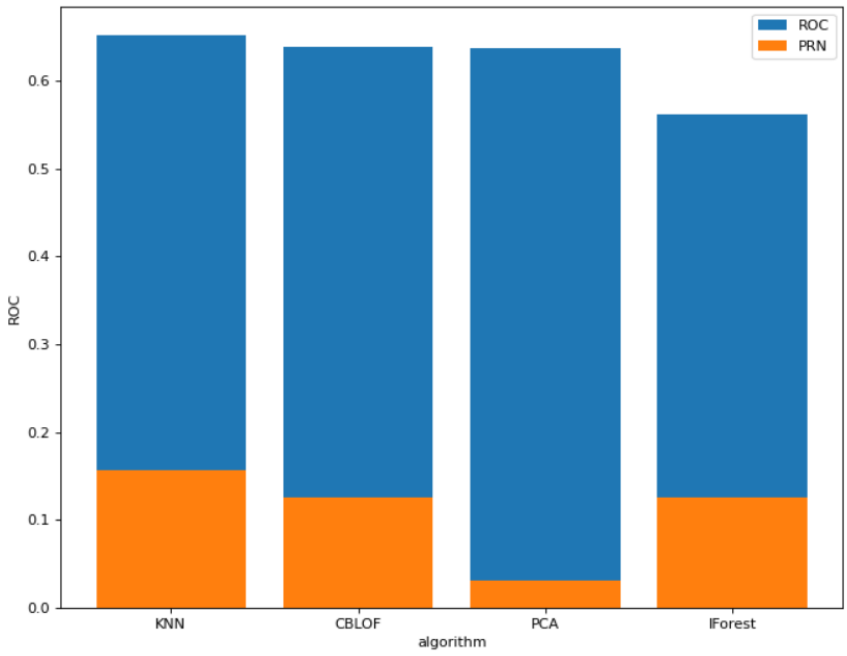
该数据集共包括 1725 个.csv 数据文件（注：文件的最大标号为 1800，但实际只包括了 1725 个文件），其中各个数据文件中共有的公共列有 9 个，分别为：

columns = ['V2', 'diff.score', 'original.label', 'V1', 'V4', 'V3', 'V7', 'V5', 'V6']

训练数据集、测试数据集在不同算法下的 ROC 值和 PRN 值如下所示：

```
On train Data:
knn ROC:0.6899, precision @ rank n:0.2577
On Test Data:
knn ROC:0.6515, precision @ rank n:0.1562
On train Data:
cblof ROC:0.6802, precision @ rank n:0.2784
On Test Data:
cblof ROC:0.6382, precision @ rank n:0.125
On train Data:
pca ROC:0.6925, precision @ rank n:0.2474
On Test Data:
pca ROC:0.6374, precision @ rank n:0.0312
On train Data:
iforest ROC:0.6398, precision @ rank n:0.2268
On Test Data:
iforest ROC:0.5614, precision @ rank n:0.125
KNN average ROC: 0.6515
```

我们将各算法在测试集上的实验结果进行可视化对比，得到如下对比图：



从图中我们可以发现，在 ROC 值方面，KNN 算法的实验结果最佳，IForest 算法结果最差；在 PRN 值上，同样是 KNN 算法的实验结果最佳，但结果最差的却是 PCA 算法。