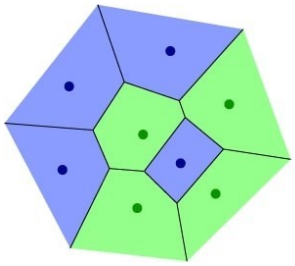

Digital twin k-NN-based accuracy control

This code allows monitoring digital twin accuracies using k-NN classifiers. It carries out the following steps:

1. Import real and digital twin datasets from excel files.
2. Scale data using Robust Scaler.
3. Classify real and digital twin data using k-NN and cross-validation (and defines the optimal number of neighbors, i.e., 'k')
4. Calculate p-chart control limits for k-NN accuracies and check for out-of-control points.
5. Plot p-chart for k-NN accuracies.

Rationale:

We fit k-NN classifiers to separate real and digital twin observations. The more accurate the digital twin, the more difficult it will be for the k-NN to separate both groups of observations. If the null hypothesis (H0: real and digital twin data come from the same distribution) is true, the best decision k-NN can take is to classify each observation at random. In this case, k-NN accuracies will remain around 50% (for balanced real/digital twin datasets). However, if real and digital twin data come from totally different distributions, the k-NN classification accuracy may reach up to 100%. So it is important to keep in mind that a k-NN accuracy of 50% indicates a very accurate digital twin, while a k-NN accuracy of 100% indicates a very innacurate digital twin.

Main parameters:

1. **base_obs:** number of groups that will be used for estimation of control limits.
2. **real_data_file**: name of the real data workbook, with extension.
3. **dt_data_file**: name of the digital twin data workbook, with extension.

If these workbooks are in the same directory as the current code, real_data_file and dt_data_file may just refer to the workbooks' names.

Requirements:

1. Datasets from excel files must contain observations as rows and observations' characteristics as columns. Both univariate and multivariate datasets are acceptable.
2. Separate workbooks for real and digital twin are required.
3. Datasets must contain a group column named '_group' (i.e., the control chart group index as an integer value).
4. Only numeric data types are allowed: integer, real, boolean (0/1), or already encoded categorical data (e.g., label encoded, one-hot encoded).

5. Column names other than '_group' may not start with underscore.
6. Choose adequate group sizes for a suitable chance of detecting an accuracy shift. For group size planing, please refer to '*Introduction to Statistical Quality Control*' (Montgomery, 2005).

In [1]:

```
# Import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statistics
from sklearn.preprocessing import RobustScaler, MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
import math
from scipy import stats
```

In [2]:

```
def read_data(real_data_file, dt_data_file=None):
    # Real data
    real_data = pd.read_excel(real_data_file, engine='openpyxl')
    real_data['_type'] = 'real data'
    # Digital twin data
    dt_data = pd.read_excel(dt_data_file, engine='openpyxl')
    dt_data['_type'] = 'digital twin'
    return real_data, dt_data
```

In [3]:

```
def classifier_test(all_data):
    # Define the base number of neighbors
    n_neighbors = int(np.ceil(math.sqrt(all_data.shape[0])) // 2 * 2 + 1)
    # Define pipeline for scaling
    pipe = Pipeline([("scale", RobustScaler()),
                     ("knn", KNeighborsClassifier())
                    ])
    # Extract X and y
    X = all_data.loc[:, ~all_data.columns.str.startswith('_')].values
    y = all_data['_type'].values.ravel()
    # Grid search for the number of neighbors
    param_grid = {
        'knn__n_neighbors': [int(np.ceil(n_neighbors*i/4) // 2 * 2 + 1)
                             for i in range(1,9)]
    }
    search = GridSearchCV(pipe, param_grid, n_jobs=-1, cv=5)
    # Train
    search.fit(X, y)
    # Return accuracy and subgroup size
    return [search.best_score_, all_data.shape[0]]
```

In [4]:

```
def plot_p_chart(p, plot):
    # Number of groups to plot
    n_groups = p.shape[0]
    # Initialize array of out-of-control points
    OCPs = np.zeros(n_groups)
    # Initialize array of point colors
    colors = []
    # Define base observations
    p_base = p.iloc[:base_obs]

    # Calculate control limits
    UCL = np.mean(p_base['p'])+3*(np.sqrt((np.mean(p_base['p'])*
                (1-np.mean(p_base['p'])))/np.mean(p_base['group_size'])))
    LCL = np.mean(p_base['p'])-3*(np.sqrt((np.mean(p_base['p'])*
                (1-np.mean(p_base['p'])))/np.mean(p_base['group_size'])))
    # Check out-of-control points
    for idx, row in p.iterrows():
        if (row['p'] < LCL or row['p'] > UCL) and idx >= base_obs:
            colors.append('red')
            OCPs[idx] = 1
        else:
            colors.append('C0')

    # Plot control chart
    if (plot):
        plt.figure(figsize=(15,7.5))
        plt.plot(p['p'], linestyle='-', zorder = 1)
        plt.scatter(x=range(0,len(p['p'])), y=p['p'], c=colors, zorder = 5)
        plt.step(x=range(0,len(p['p'])), y=n_groups*[UCL], color='red',
                 linestyle='dashed', zorder = 10)
        plt.step(x=range(0,len(p['p'])), y=n_groups*[LCL], color='red',
                 linestyle='dashed', zorder = 10)
        plt.axhline(statistics.mean(p_base['p']), color='black', linewidth=0.7)
        plt.xlabel('Group')
        plt.ylabel('k-NN accuracy')
        plt.savefig('control_chart.jpg', format='jpg', dpi=1200)
        plt.savefig('control_chart.svg', format='svg', dpi=1200)

    return OCPs
```

In [5]:

```
def evaluate_data(real_data, dt_data, plot=True):
    # Initialize array of accuracies
    accuracies = []
    # Classify group data using k-NN
    for group in real_data['_group'].unique():
        real_sample = real_data[real_data['_group'] == group]
        dt_sample = dt_data[dt_data['_group'] == group]
        both_samples = pd.concat([real_sample, dt_sample])
        accuracies.append(classifier_test(both_samples))
    # k-NN output: probabilities and group sizes
    p = pd.DataFrame(accuracies, columns=["p", "group_size"])
    # Plot control chart and return out-of-control points
    OCPs = plot_p_chart(p, plot=plot)
    return OCPs
```

In [6]:

```
# Run control chart
base_obs = 25
real_data, dt_data = read_data(real_data_file='PS data.xlsx',
                               dt_data_file='DT data.xlsx')
OCPs = evaluate_data(real_data=real_data.dropna(), dt_data=dt_data.dropna())
```