1. Nested cross-validation exercise

Nested cross-validation for feature selection with nearest neighbors

- Use Python 3 to program both a hyper-parameter selection method based on 5-fold cross-validation and a nested 5-fold cross-validation for estimating the prediction performance of models inferred with this automatic selection approach. Use base learning algorithm provided in the exercise, namely the "use_ith_feature" method, so that the value of the hyper-parameter i is automatically selected from the range from 1 to 100 of alternative values. The provided base learning algorithm "use_ith_feature" is 1-nearest neighbor that uses only the ith feature of the data given to it. The 5-fold CV based hyper-parameter selection procedure is supposed to select the best feature, e.g. the value of i, based on C-index evaluated with predictions obtained with 5-fold cross-validation. A ready-made implementation of C-index is also provided in the exercise. In nested 5-fold cross-validation, a C_index value is further evaluated on the predictions obtained from an outer 5-fold cross-validation. During each round of this outer 5-fold CV, the whole feature selection process based on inner 5-fold CV is separately done and the selected feature is used for prediction for the test data points held out during that round of the outer CV. Accordingly, the actual learning algorithm, whose prediction performance will be evaluated with nested CV, is the one that automatically selects the single best feature with 5-fold cross-validation based model selection (see the lectures and the pseudo codes presented on them for more info on this interpretation).
- Compare the C-index produced by nested 5-fold CV with the result of ordinary 5-fold CV with the best value of i e.g. the feature providing the highest 5-fold CV C-index, and show the C-index difference between the two methods.
- Use the provided implementation of the "use_ith_feature" learning algorithm and C-index functions in your exercise.

As a summary, for completing this exercise implement the following steps:

---

1. Use 5-fold cross-validation for determining the optimal i-parameter for the data (X_train.csv, y_prediction.csv) from the set of possible values of i e.g. {1,...,100}. When you have found the optimal i, save the corresponding C-index (call it 5_fold_c_index) for this parameter.

2. Similarly, use nested cross-validation (5-fold CV both in outer and inner folds) for estimating the C-index (call it n_5_fold_c_index) of the method that selects the best feature with 5-fold approach.

3. Return both this notebook and as a PDF-file made from it in the exercise submit page.

Remember to use the provided learning algorithm use_ith_feature and C-index functions in your implementation!

Import libraries

In [7]:
```
#In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
```

Provided functions

In [8]:
```
"""
C-index function:
- INPUTS:
'y' an array of the true output values
'yp' an array of predicted output values
- OUTPUT:
The c-index value
"""
def cindex(y, yp):
    n = 0
    h_num = 0
    for i in range(0, len(y)):
        t = y[i]
        p = yp[i]
        for j in range(i+1, len(y)):
            nt = y[j]
            np = yp[j]
            if (t != nt):
                n = n + 1
                if (p < np and t < nt) or (p > np and t > nt):
                    h_num += 1
                elif (p == np):
                    h_num += 0.5
    return h_num/n

"""
Self-contained 1-nearest neighbor using only a single feature
- INPUTS:
'X_train' a numpy matrix of the X-features of the train data points
'y_train' a numpy matrix of the output values of the train data points
'X_test' a numpy matrix of the X-features of the test data points
'i' the index of the feature to be used with 1-nearest neighbor
- OUTPUT:
'y_predictions' a list of the output value predictions
"""
def use_ith_feature(X_train, y_train, X_test, i):
    y_predictions = []
    for test_ind in range(0, X_test.shape[0]):
        diff = X_test[test_ind, i] - X_train[:, i]
        distances = np.sqrt(diff * diff)
        sort_inds = np.array(np.argsort(distances), dtype=int)
        y_predictions.append(y_train[sort_inds[0]])
    return y_predictions
```

Your implementation here

In [9]:
```
# In this cell implement the required tasks
# Read the csv files, data dose not contain headers(column names).
# Dimension of X_train.csv is (30, 100) and for y_prediction.csv is (30, 1)

# Loading the feature matrix X_train and target variable y_prediction
x_train = pd.read_csv(r'C:\Users\Comp-servis\Desktop\Evaluation of Machine Learning Meth
y_prediction = pd.read_csv(r'C:\Users\Comp-servis\Desktop\Evaluation of Machine Learning

# Converting the data to numpy arrays
X_train = x_train.values
```

```
y_train = y_prediction.values.squeeze()

# The resulting X_train and y_prediction arrays will be used for Machine Learning tasks
```

In [10]:

```
# Defining the select_hyperparameter function that selects the optimal i-parameter and c
def select_hyperparameter(X, y, kf):

    # Variables to store the optimal i-parameter and corresponding C-index
    best_i = -1
    best_c_index = 0

    # Iterating from 0 till 100
    for i in range(0, 100):

        # A list to store all the C-index scores in each fold
        c_index_scores = []

        # Performing 5-fold cross-validation using the provided KFold object (kf)
        for train_index, val_index in kf.split(X):
            X_train_fold, X_val_fold = X[train_index], X[val_index]
            y_train_fold, y_val_fold = y[train_index], y[val_index]

            # Using the provided use_ith_feature function for predictions
            y_pred = use_ith_feature(X_train_fold, y_train_fold, X_val_fold, i)

            # Determining C-index for the current fold
            score = cindex(y_val_fold, y_pred)

            # Appending the result to the list
            c_index_scores.append(score)

        # Calculating the average C-index for the current hyperparameter
        avg_c_index = np.mean(c_index_scores)

        # Updating the best hyperparameter in the case of a higher C-index
        if avg_c_index > best_c_index:
            best_c_index = avg_c_index
            best_i = i

    # Returning the best hyperparameter and its corresponding C-index
    return best_i, best_c_index

# Defining the nested_cv function for the nested cross-validation method
def nested_cv(X, y):

    # A list to store C-index scores for the outer folds
    outer_scores = []

    # Creating a 5-fold cross-validation object for the outer loop
    outer_kf = KFold(n_splits=5)

    # Iterating over the outer folds
    for train_index, test_index in outer_kf.split(X):

        # Spliting the data into training and testing sets for the outer fold
        X_train_fold, X_test_fold = X[train_index], X[test_index]
        y_train_fold, y_test_fold = y[train_index], y[test_index]

        # Creating a 5-fold cross-validation object for the inner loop
        inner_kf = KFold(n_splits=5)

        # Determining the best hyperparameter using the inner loop
        best_i, _ = select_hyperparameter(X_train_fold, y_train_fold, inner_kf)
        # Using the best hyperparameter to make predictions on the test set of the outer
        y_pred = use_ith_feature(X_train_fold, y_train_fold, X_test_fold, best_i)

        # Calculating C-index for the predictions
        score = cindex(y_test_fold, y_pred)

        # Appending the result to the list
        outer_scores.append(score)

    # Returning the mean C-index over all outer folds
    return np.mean(outer_scores)

# Obtaining values for the optimal i-parameter and corresponding c-index
best_i, five_fold_c_index = select_hyperparameter(X_train, y_train, KFold(n_splits=5, sh

# The C-index of the method that selects the best feature in the nested cross-validation
n_5_fold_c_index = nested_cv(X_train, y_train)

print('The optimal i-parameter is', best_i)

print('The corresponding C-index value for the optimal i-parameter in the cross-validati

print('The C-index of the method that selects the best feature in the nested cross-valid

print('The difference between the C-indexes is', five_fold_c_index - n_5_fold_c_index)
```

```
The optimal i-parameter is 19
The corresponding C-index value for the optimal i-parameter in the cross-validation is
0.7066666666666667
The C-index of the method that selects the best feature in the nested cross-validation i
s 0.5599999999999999
The difference between the C-indexes is 0.14666666666666672
```

The C-index value for cross-validation is higher than nested cross-validation, which means that the performance of the cross-validation is better than that of the nested cross-validation. This interesting situation can be explained with the fact that nested cross-validation method does not perform well in the small datasets. Nested cross-validation introduces unnecessary complexity in the small datasets, which leads to high variance in performance estimates.