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 [1]:
```
#In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd # for data manipulation

from sklearn.model_selection import KFold # for cross-validation
```

Provided functions

In [2]:
```
"""
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 [3]:
```
# In this cell implement the required tasks
# Read the csv files, data does not contain headers(column names).
# Dimension of X_train.csv is (30, 100) and for y_prediction.csv is (30, 1)
```

```
#Let's first read our data.
train_data = pd.read_csv("X_train.csv", header=None)
predict_data = pd.read_csv("y_prediction.csv", header=None)

#Let's turn our dataframes into numpy arrays so they split correctly
train_data = train_data.to_numpy()
predict_data = predict_data.to_numpy()

#Initializing our k-folds
kfold = KFold(n_splits=5, shuffle=True, random_state=1)

#Splitting training data into train and test Source https://machinelearningmastery.com/t
for train, test in kfold.split(train_data, predict_data):
    X_train, X_test = train_data[train], train_data[test]
    y_train, y_test = predict_data[train], predict_data[test]


#Let's create an empty dictionary to save our c-indexes by i values
c_to_i = {}

#Let's then iterate through our all i values from 1 to 100, use those in calculating the

for i in range (1,100):
    y_pred = use_ith_feature(X_train, y_train, X_test, i)
    c_value = cindex(y_test, y_pred)
    c_to_i.update({i : c_value})

#Now we can find our most optimal C-index, which is the largest one. Easy way is to crea
c_sorted = sorted(c_to_i.items(), key=lambda x:x[1])

#Last one is that the index of length minus one.
#So best i value is 19 with a C of 0.96...
five_fold_c_index = c_sorted[len(c_sorted)-1][1]
print(five_fold_c_index)
```

```
0.9666666666666667
```

By using the 5-fold cross-validation, we got the best i value as 19 with a c value of 0.966...

In [4]:

```
#Now let's utilize nested cross-validation by starting with the outer fold.
outer_kfold = KFold(n_splits=5, shuffle=True, random_state=1)

#Empty dictionary to save our c-indexes by i values
c_to_i = {}

#Utilizing the earlier split, but this time we do it twice as both outer and inner versi
for train, test in outer_kfold.split(train_data, predict_data):
    X_train, X_test = train_data[train], train_data[test]
    y_train, y_test = predict_data[train], predict_data[test]

    inner_kfold = KFold(n_splits=5, shuffle=True, random_state=1)

    for i in range(1,100):
        for inner_train, inner_test in inner_kfold.split(X_train, y_train):
            X_inner_train, X_inner_test = X_train[inner_train], X_train[inner_test]
            y_inner_train, y_inner_test = y_train[inner_train], y_train[inner_test]

            y_inner_pred = use_ith_feature(X_inner_train, y_inner_train, X_inner_test, i
            c_value = cindex(y_inner_test, y_inner_pred)
            c_to_i.update({i : c_value})

#find our most optimal C-index, which is the largest one. Easy way is to crea
```

```
c_sorted = sorted(c_to_i.items(), key=lambda x:x[1])

#Last one is that the index of length minus one.
#So best i value is 29 with a C of 1
n_five_fold_c_index = c_sorted[len(c_sorted)-1][1]
print(n_five_fold_c_index)
```

```
1.0
```

By using the nested 5-fold cross-validation, we find out that the best i value is 29 with a C value of 1.