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

Provided functions

In [ ]:
```
"""
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 [ ]:

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

# read X_train
X = pd.read_csv('X_train.csv',
                header=None).to_numpy()

# read y_prediction
y = pd.read_csv('y_prediction.csv',
                header=None).to_numpy()


# kfold with shuffling
kf_shuffle = KFold(n_splits=5, shuffle=True)

# kfold without shuffling
```

```
kf = KFold(n_splits=5, shuffle=False)



### 5-fold cross-validation

five_fold_c_index = 0 #set starting best c-index
best_i = -1 #set starting i

# iterate through folds
for a, (train_index, test_index) in enumerate(kf.split(X)):

    # find best i and corresponding c-index
    for i in range(X.shape[1]):
        y_predictions = use_ith_feature(X[train_index, :], y[train_index], X[test_index, :], i)
        c = cindex(y[test_index], y_predictions)
        if c > five_fold_c_index:
            five_fold_c_index = c
            best_i = i

# print best c-index and i
print(f"The best c-index is: {five_fold_c_index}.")
print(f"The best i is: {best_i}.")

#note
print("\nNote that I shuffled the data before splitting it to folds\nto ensure the stratification of the data set. This change
```

```
The best c-index is: 1.0.
The best i is: 18.

Note that I shuffled the data before splitting it to folds
to ensure the stratification of the data set. This changes
the results every run. Without the shuffle the best i is 18
with c-index 1.0
```

In [ ]:

```
### nested 5-fold cross-validation

def nested_cv(X, y):

    n_5_fold_c_index = 0
    five_fold_c_index = 0 #set starting best c-index
```

```
    inner_best_i = -1 #set starting i
    outer_best_i = -1 #set starting overall i

    # iterate through folds
    for a, (outer_train_index, outer_test_index) in enumerate(kf_shuffle.split(X)):
        X_train = X[outer_train_index]
        y_train = y[outer_train_index]
        # inner loop
        for b, (inner_train_index, inner_test_index) in enumerate(kf.split(X_train)):

            # find best i and corresponding c-index
            for i in range(X.shape[1]):
                y_predictions = use_ith_feature(X_train[inner_train_index, :], y_train[inner_train_index], X_train[inner_test_
                c = cindex(y_train[inner_test_index], y_predictions)
                if c > five_fold_c_index:
                    five_fold_c_index = c
                    inner_best_i = i

        # outer loop (using only the best i found in the inner loop we can estimate the performance of the model more reliably
        y_predictions = use_ith_feature(X[outer_train_index, :], y[outer_train_index], X[outer_test_index, :], inner_best_i)
        c = cindex(y[outer_test_index], y_predictions)
        if c > n_5_fold_c_index:
            n_5_fold_c_index = c
            outer_best_i = inner_best_i
    return n_5_fold_c_index, outer_best_i

# print best c-index and i
print(f"The best c-index is: {nested_cv(X, y)[0]}.")
print(f"The best i is: {nested_cv(X, y)[1]}.")

#note
print("\nNote that I shuffled the data before splitting it to folds\nto ensure the stratification of the data set. This change
print("\nWe can get an average c-index by running the model multiple times with shuffle on.")

nested_c_list = []
for x in range(100):
    nested_c_list.append(nested_cv(X, y)[0])

average_c = np.mean(nested_c_list)
print(f"Estimation of c-index from a hundred runs: {average_c}")
```

```
The best c-index is: 0.9.
The best i is: 26.

Note that I shuffled the data before splitting it to folds
to ensure the stratification of the data set. This changes
the results every run. Without the shuffle the best i is 76
with c-index 0.8

We can get an average c-index by running the model multiple times with shuffle on.
Estimation of c-index from a hundred runs: 0.7246666666666667
```