In [1]:

```
## Student info
__name__ = 'Jeremias Shadbolt'
__utumail__ = 'jrshad@utu.fi'
__student_no__ = '1900386'
```

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 leave-one-out cross-validation and a nested leave-one-out 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 LOOCV 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 leave-one-out cross-validation. A ready-made implementation of C-index is also provided in the exercise. In nested leave-one-out cross-validation, a C_index value is further evaluated on the predictions obtained from an outer leave-one-out cross-validation. During each round of this outer LOOCV, the whole feature selection process based on inner LOOCV is separately done and the selected feature is used for prediction for the test data point held out during that round of the outer LOOCV. 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 leave-one-out cross-validation based model selection (see the lectures and the pseudo codes presented on them for more info on this interpretation).
- Note that since the hold-out set in LOOCV has only a single datum but C-index requires at least two data points. The solution in this exercise is to "pool" the predictions of all LOOCV rounds of a single LOOCV computation into an array of length of the data used in that LOOCV computation and then compute C-index on that array and the corresponding true outputs. This pooling approach, however, does have its weaknesses, since C-index computed from pooled LOOCV outputs may sometimes be a heavily biased estimator of the true C-index. This has been considered in detail in our previous research (and other group's too as seen in the references) that is available here: http://dx.doi.org/10.1177/0962280218795190 where AUC, a special case of C-index, is considered. The study goes quite deep into the problem of AUC estimation with CV, and you can read it if you are interested about the research carried out in our laboratory, while EMLM course does not go that far and this year's exercise unfortunately still has this non-optimal pooling approach in use.
- Compare the C-index produced by nested leave-one-out CV with the result of ordinary leave-one-out CV with the best value of i e.g. the feature providing the highest LOOCV 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 leave-one-out cross-validation for determining the optimal i-parameter for the data (X_alternative.csv, y_alternative.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 loo_c_index) for this parameter.

2. Similarly, use nested leave-one-out cross-validation (leave-one-out both in outer and inner folds) for estimating the C-index (call it nloo_c_index) of the method that selects the best feature with leave-one-out 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 [2]:

```
#In this cell import all libraries you need. For example:
import numpy as np
from sklearn.model_selection import LeaveOneOut
```

In [3]:

```
# get data
y_data = np.genfromtxt('y_alternative.csv', delimiter=',', dtype = np.float64)
x_data = np.genfromtxt('X_alternative.csv', delimiter=',', dtype = np.float64)
```

In [4]:

```
# verify
print(f"x_data: {x_data[:1]}")
print(f"y_data: {y_data[:5]}")

print(f"y_data shape: {y_data.shape}")
print(f"x_data shape: {x_data.shape}")
```

```
x_data: [[0.61934953 0.89439371 0.52379425 0.38570093 0.40956954 0.48489574
  0.07507832 0.45906914 0.62770591 0.68823611 0.15247277 0.28527357
  0.35032215 0.25327082 0.07629901 0.95850396 0.38517872 0.07808428
  0.5075518  0.27236985 0.76152001 0.15095591 0.30428641 0.47423309
  0.65460813 0.68834395 0.65755344 0.36938033 0.17470625 0.08606935
  0.81995635 0.74082992 0.70136535 0.76704102 0.76472048 0.00549333
  0.62417165 0.23090696 0.76890586 0.92275529 0.07735633 0.7583811
  0.08242645 0.92603414 0.69926603 0.83237585 0.82842544 0.98624802
  0.81411375 0.23637566 0.31982617 0.97585136 0.35965544 0.09743788
  0.83479166 0.52893187 0.58661422 0.95711808 0.20533743 0.9315812
  0.42919528 0.30204538 0.56669822 0.62289021 0.60407256 0.25046648
  0.09030286 0.20138562 0.93401687 0.57705361 0.85291432 0.99177122
  0.64068785 0.20521913 0.29412972 0.09797216 0.94063162 0.83614917
  0.89299978 0.75235432 0.42126456 0.0203019  0.16223335 0.94407351
  0.49388677 0.86148023 0.84636966 0.65876625 0.47844355 0.01581105
  0.25640376 0.25598754 0.83907536 0.68569073 0.73768221 0.94996171
  0.05162623 0.49494866 0.41641368 0.78041606]]
y_data: [0.36394716 0.65434878 0.37008345 0.83379345 0.19850295]
y_data shape: (30,)
x_data shape: (30, 100)
```

Provided functions

In [5]:

```
"""
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

1. Use leave-one-out cross-validation for determining the optimal i-parameter for the data (X_alternative.csv, y_alternative.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 loo_c_index) for this parameter.

In [41]:

```
# Although this is incorrectly named as to only give the best i-parameter it also provides
def get_best_i_parameter(x_data, y_data):
    # Leave-one-out-CV
    loocv = LeaveOneOut()

    # list for accuracies of different cycles
    accuracies = dict()

    # while-loop to go through feature indexes
    # append predictions and actual values per each loop of LOOCV to dictionary

    feature_idx = 0
    while feature_idx <= 99:
        preds = list()
        actuals = list()
        for train_idx, test_idx in loocv.split(x_data):
            x_train, x_test = x_data[train_idx], x_data[test_idx]
            y_train, y_test = y_data[train_idx], y_data[test_idx]
            preds.append(use_ith_feature(x_train, y_train, x_test, feature_idx))
            actuals.append(y_test)
        accuracies[feature_idx] = cindex(actuals, preds)
        feature_idx += 1
    loo_c_index = max(accuracies.values())
    index = max(accuracies, key = accuracies.get)
    return loo_c_index, index
```

In [43]:

```
# get best feature based on C-index

print(f"The best C-Index was {get_best_i_parameter(x_data, y_data)[0]:.2f} for feature inc
```

```
The best C-Index was 0.66 for feature index 76
```

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

In [63]:

```
# Nested LeaveOneOut Cross Validation
def nloocv(x_data, y_data):
    loocv = LeaveOneOut()
    preds = list()
    actuals = list()

    # outer loocv
    for train_idx, test_idx in loocv.split(x_data):
        x_train, x_test = x_data[train_idx], x_data[test_idx]
        y_train, y_test = y_data[train_idx], y_data[test_idx]

        #inner loocv
        c_index, best_i = get_best_i_parameter(x_train, y_train)

        #predictions on subset with selected i-feature
        prediction = use_ith_feature(x_train, y_train, x_test, best_i)
        preds.append(prediction)
        actuals.append(y_test)

    # get c_index for iterations
    n_loo_cindex = cindex(actuals, preds)
    return n_loo_cindex
```

In [64]:

```
# compare results
print(f"C-index from nested LOOCV: {nloocv(x_data, y_data):.2f}")
print(f"C-index from regular LOOCV: {get_best_i_parameter(x_data, y_data)[0]:.2f}")
```

```
C-index from nested LOOCV: 0.51
C-index from regular LOOCV: 0.66
```