# 1. Nested cross-validation exercise


- Lauri Orava
- 516761
- laorav@utu.fi
- (Cooperation with Milja Lempinen)


## Nested cross-validation for k-nearest neighbors

- Use Python 3 to program a nested leave-one-out cross-validation for the k-nearest neighbors (kNN) method so that the number of neighbours k is automatically selected from the set k = [3, 5, 7, 9, 11]. In other words, the base learning algorithm is kNN but the actual learning algorithm, whose prediction performance will be evaluated with nested CV, is kNN with automatic CV-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 leave-one-out CV with normal leave-one-out cross-validation with the best value of k.
- As a kNN implementation, use the provided kNN and C-index functions in your exercise.
- Use the CV implementations on the provided subsampled iris data (100 randomly drawn data points from iris) and report the resulting classification accuracy via C-index. Hint: you can use the nested CV example provided on sklearn documentation: https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html as a starting point, but do NOT use the ready made CV implementations of sklearn.

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

---

**1. Use leave-one-out cross-validation for determining the optimal k-parameter for the data (X.csv, y.csv) from the set k = [3,5,7,9,11]. When you have solved the optimal k-parameter, save the corresponding C-index (call it loo_c_index) for this best value of k.**

**2. Similarly, use nested leave-one-out cross-validation (leave-one-out both in outer and inner folds) for determining the C-index (call it nloo_c_index) of the kNN + leave-one-out cross-validation based k selection approach.**

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

---

Remember to use the provided C-index and kNN functions in your implementation!

## Import libraries and data

In [ ]:

```
#In this cell import all libraries you need. For example:
import numpy as np
import pandas as pd
import sklearn.model_selection
import matplotlib.pyplot as plt

X = pd.read_csv('data/X.csv').to_numpy()
y = pd.read_csv('data/y.csv').to_numpy()
```

## 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 k-nearest neighbor
- 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
'k' the k-parameter integer value for kNN
- OUTPUT:
'y_predictions' a list of the output value predictions
"""
def knn(X_train, y_train, X_test, k):
    y_train = np.array(y_train, dtype=int)
    y_predictions = []
    for test_ind in range(0, X_test.shape[0]):
        diff = X_test[test_ind, :].reshape(1, -1) - X_train
        distances = np.sqrt(np.sum(diff * diff, axis = 1))
        sort_inds = np.array(np.argsort(distances), dtype=int)
        counts = np.bincount(y_train[sort_inds[0:k]])
        y_predictions.append(np.argmax(counts))
    return y_predictions
```

# Your implementation here

## Determine optimal k-parameter using LOOCV

In [ ]:
```
def find_best_k(X, y, loo):
    kSet = [3, 5, 7, 9, 11]
    loo_c_index = 0
    # Try for every k in our parameter set
    for k in kSet:

        y_preds = []

        for train_i, test_i in loo.split(X):

            # Prepare train and test datasets
            X_train, X_test = X[train_i], X[test_i]
            y_train, y_test = y[train_i].reshape(-1), y[test_i].reshape(-1)

            # Calculate kNN predictions using the given function
            y_preds.append(knn(X_train, y_train, X_test, k))
```

```
        # Calculate C-index with the given function
        c_index = cindex(y, y_preds)
        #print("For k-parameter " + str(k) + " C-index is " + str(c_index))

        # Replace optimal k-parameter and C-index if necessary
        if c_index >= loo_c_index:
            loo_c_index = c_index
            best_k = k

    #print("Optimal k-parameter is " + str(best_k) + " where C-index is " + str(loo_c_index))
    return best_k, loo_c_index

loo = sklearn.model_selection.LeaveOneOut()
find_best_k(X, y, loo)
```

Out[ ]:
```
(7, 0.9742804654011022)
```

## Determine C-index of previous selection using NLOOCV

In [ ]:
```
# Prepare variables
loo_outer = sklearn.model_selection.LeaveOneOut()
nloo_c_index_list = []

# Start outer loop
for train_ix, test_ix in loo_outer.split(X):

    # Split into training and test sets
    X_train, X_test = X[train_ix, :], X[test_ix, :]
    y_train, y_test = y[train_ix].reshape(-1), y[test_ix].reshape(-1)

    # Prepare inner leave-one-out loop
    loo_inner = sklearn.model_selection.LeaveOneOut()

    # Use previous function to find C-index values
    best_nk, nloo_c_index = find_best_k(X_train, y_train, loo_inner)
    nloo_c_index_list.append(nloo_c_index)

print("C-index average of LOOCV: ", np.mean(nloo_c_index_list))
```

```
C-index average of LOOCV:  0.9745928694670772
```