1. Nested cross-validation exercise

- Milja Lempinen
- 520600
- mmlemp@utu.fi
- This exercise was done partly in cooperation with Lauri Orava

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

In [1]:

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

Provided functions

In [2]:

```python
"""
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
    try:
        return h_num/n
    except ZeroDivisionError:
        return h_num/0.5

"""
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
        #Modified this, cause of object issues
        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

In [3]:

```python
# In this cell implement the required tasks with comments on the code.
#Reading the csv-files into pandas dataframes
X_path = 'X.csv'
y_path = 'y.csv'
X = pd.read_csv(X_path).to_numpy()
y = pd.read_csv(y_path).to_numpy(dtype=int)
```

In [4]:
```python
#loo-cv
"""
Get indices for leave one out -cross validation
- INPUTS:
'X' the feature matrix
'y' the labels
'n' number of loo iterations
- OUTPUT:
'train_indices' an array of indices for the train-data
'test_indices' an array of indices for the test-data
"""
def loo_split(X):
    X_indices = np.arange(0, len(X))
    test_indices = []
    train_indices = []
    for i in range(0, len(X)):
        test_indices.append(i)
        train_indices.append(X_indices[np.arange(len(X))!=i])
    return(train_indices, test_indices)
```

In [5]:
```python
#The set of possible k-values to be tested
k = [3,5,7,9,11]
```

In [6]:
```python
#leave-one-out cross-validation with own loo_split
def get_best_k_orig(X, y, k, loosplit):
    t0 = time.time()
    loo_c_index = 0
    best_k = 0

    for i in k:
        yp = []
        for j in range(0, len(X)):
            X_train = X[loosplit[0][j]]
            X_test = X[loosplit[1][j]].reshape(1, 4)
            y_train = y[loosplit[0][j]].flatten()
            y_test = y[loosplit[1][j]].flatten()

            yp.append(knn(X_train, y_train, X_test, i))
            #print(model)

        c_index = cindex(y, yp)

        if c_index >= loo_c_index:
            loo_c_index = c_index
            best_k = i

        #print('For k:' + str(i) + ' C-index is ' +str(c_index))

    #print('Best C-index is ' + str(loo_c_index) + ' where k is ' + str(be
    t1 = time.time()
    time_total = t1-t0
    return(best_k, loo_c_index, time_total)

loosplit = loo_split(X)
get_best_k_orig(X, y, k, loosplit)
```

Out[6]: (7, 0.9742804654011022, 0.06521773338317871)

In [7]:
```
#leave-one-out cross-validation with sklearn loo implementation
def get_best_k(X, y, k, model):
    t0 = time.time()
    loo_c_index = 0
    best_k = 0

    for i in k:
        yp = []
        for train_i, test_i in model.split(X):
            X_train, X_test = X[train_i], X[test_i]
            y_train, y_test = y[train_i].reshape(-1), y[test_i].reshape(-1

            yp.append(knn(X_train, y_train, X_test, i))
            #print(model)

        c_index = cindex(y, yp)

        if c_index >= loo_c_index:
            loo_c_index = c_index
            best_k = i

        #print('For k:' + str(i) + ' C-index is ' +str(c_index))

    #print('Best C-index is ' + str(loo_c_index) + ' where k is ' + str(be
    t1 = time.time()
    time_total = t1-t0
    return(best_k, loo_c_index, time_total)

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

Out[7]:
```
(7, 0.9742804654011022, 0.06511998176574707)
```

In [8]:

```
#nested leave-one-out cross-validation using the sklearn loo-cv -objects

nloo_c_index_l = []
best_k = 0
time_0 = time.time()
for i in k:
    inner_cv = sklearn.model_selection.LeaveOneOut()
    outer_cv = sklearn.model_selection.LeaveOneOut()
    yp = []

    #outer loop train-test sets
    for train_i, test_i in outer_cv.split(X):
        X_train, X_test = X[train_i], X[test_i]
        y_train, y_test = y[train_i].reshape(-1), y[test_i].reshape(-1)

        #inner loop
        best_ik, nloo_c_index, inner_time = get_best_k(X_train, y_train, k
        nloo_c_index_l.append(nloo_c_index)




        yp.append(knn(X_train, y_train, X_test, i))
        #print(model)

    #print('For k:' + str(i) + ' C-index is ' +str(c_index))
time_1 = time.time()
time_total = time_1-time_0

print('C-index average is ' + str(np.mean(nloo_c_index_l)))
print('This took ' + str(time_total) + ' seconds.')
```

```
C-index average is 0.9745928694670772
This took 15.181504964828491 seconds.
```

In [9]:

```
#With own loo-implementation

nloo_c_index_l = []
best_k = 0
time_0 = time.time()
for i in k:
    outer_cv_split = loo_split(X)
    yp = []

    #outer loop train-test sets
    for train_i, test_i in outer_cv.split(X):
        X_train, X_test = X[train_i], X[test_i]
        y_train, y_test = y[train_i].reshape(-1), y[test_i].reshape(-1)

        #inner loop
        inner_cv_split = loo_split(X_train)
        best_ik, nloo_c_index, inner_time = get_best_k_orig(X_train, y_tra
        nloo_c_index_l.append(nloo_c_index)




        yp.append(knn(X_train, y_train, X_test, i))
        #print(model)

    #print('For k:' + str(i) + ' C-index is ' +str(c_index))
time_1 = time.time()
time_total = time_1-time_0

print('C-index average is ' + str(np.mean(nloo_c_index_l)))
print('This took ' + str(time_total) + ' seconds.')
```

```
C-index average is 0.9745928694670772
This took 12.41414999961853 seconds.
```

The c-index produced by the nested leave-one-out cross-validation is a little bit better than the c-index we got from only one round of cross-validation. The time it took for the notebook to process the nested leave-one-out -implementation was way longer, even for a dataset this small. It is not most likely a good choice for datasets any larger.