Lauri Reima

2109673

loreim@utu.fi

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
```

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 dose not contain headers(column names).
# Dimention of X_train.csv is (30, 100) and for y_prediction.csv is (30,
xtrain = pd.read_csv('X_train.csv', header=None)
ypred = pd.read_csv('y_prediction.csv', header=None)

## 1
def five_fold_c_index(x,y,fold): ## one can't name functions or variables
    n = len(x)
    num_folds = fold
    fold_size = n // num_folds
    indeces = np.arange(n)

    five_fold_c_index = {}

    best_index = None
    value_in_best_c_index = -1

    # loop through each i from 1...100
    for i in range(100):
        # save each i c-index
        i_fold_c_index_values = []

        # fold it five times
        for j in range(num_folds):
            # start and end indeces
            start = j * fold_size
            end = (j + 1) * fold_size

            test_ind = indeces[start:end]
            train_ind = np.concatenate([indeces[:start], indeces[end:]])

            # train and test data that changes with every fold
            X_train, X_test = x.iloc[train_ind], x.iloc[test_ind]
            y_train, y_test = y.iloc[train_ind], y.iloc[test_ind]

            # use the functions given to calculate model
            y_prediction = use_ith_feature(X_train.values, y_train.values
            c_index_val = cindex(np.array(y_test), y_prediction)
            i_fold_c_index_values.append(c_index_val)

        # save the results into a dictionary
        five_fold_c_index[i] = np.mean(i_fold_c_index_values)
        if np.mean(i_fold_c_index_values) > value_in_best_c_index:
            value_in_best_c_index = np.mean(i_fold_c_index_values)
            best_index = i

    print(f"Best Hyperparameter for {fold}-fold: {best_index} \nCorrespon
    return value_in_best_c_index
    # just to check the answer a dict was made
    # five_fold_c_index
```

In [4]:

```
def n_5_fold_c_index(x,y,fold):
    n = len(x)
    num_folds_outer = fold
    fold_size_outer = n // num_folds_outer
    indeces_outer = np.arange(n)

    nested_five_fold_c_index_outer = {}

    for i in range(100):
        # save c-index of each i
        i_fold_c_index_outer_values = []

        # do the outer folding
        for j_outer in range(num_folds_outer):
            start_outer = j_outer * fold_size_outer
            end_outer = (j_outer + 1) * fold_size_outer

            test_ind_outer = indeces_outer[start_outer:end_outer]
            train_ind_outer = np.concatenate([indeces_outer[:start_outer]

            # outer train and test data
            X_train_outer, X_test_outer = x.iloc[train_ind_outer], x.iloc
            y_train_outer, y_test_outer = y.iloc[train_ind_outer], y.iloc

            # inner loop specs
            n_inner = len(X_train_outer)
            num_folds_inner = fold
            fold_size_inner = n_inner // num_folds_inner
            indeces_inner = np.arange(n_inner)

            # save inner c-index
            i_fold_c_index_inner_values = []

            for j_inner in range(num_folds_inner):
                start_inner = j_inner * fold_size_inner
                end_inner = (j_inner + 1) * fold_size_inner

                test_ind_inner = indeces_inner[start_inner:end_inner]
                train_ind_inner = np.concatenate([indeces_inner[:start_in

                X_train_inner, X_test_inner = X_train_outer.iloc[train_in
                y_train_inner, y_test_inner = y_train_outer.iloc[train_in


                y_prediction_inner = use_ith_feature(X_train_inner.values
                c_index_val_inner = cindex(np.array(y_test_inner), y_pred

                # save the inner c-index
                i_fold_c_index_inner_values.append(c_index_val_inner)

            # save all inner c-index to outer loop
            i_fold_c_index_outer_values.append(np.mean(i_fold_c_index_inn

        # save the scores by index
        nested_five_fold_c_index_outer[i] = np.mean(i_fold_c_index_outer_
    best_index_outer = max(nested_five_fold_c_index_outer, key=nested_fiv
    best_value_outer = nested_five_fold_c_index_outer[best_index_outer]

    print(f"Best Hyperparameter for nested-{fold}-fold (Outer): {best_ind
    return best_value_outer
```

In [5]:
```
five_fold_c_index(xtrain,ypred,5)
```

```
Best Hyperparameter for 5-fold: 76
Corresponding score: 0.6533333333333334
```

Out[5]:
```
0.6533333333333334
```

In [6]:
```
n_5_fold_c_index(xtrain,ypred,5)
```

```
Best Hyperparameter for nested-5-fold (Outer): 47
Corresponding Performance (Outer): 0.7333333333333334
```

Out[6]:
```
0.7333333333333334
```

In [7]:
```
print(f"The difference in the c-index scores is: {n_5_fold_c_index(xtrain
```

```
Best Hyperparameter for nested-5-fold (Outer): 47
Corresponding Performance (Outer): 0.7333333333333334
Best Hyperparameter for 5-fold: 76
Corresponding score: 0.6533333333333334
The difference in the c-index scores is: 0.07999999999999996
```

In [ ]: