Nested_cross_validation_lkmail

January 27, 2024

1 1. Nested cross-validation exercise

1.1 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!

1.2 Import libraries

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

1.3 Provided functions

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

1.4 Your implementation here

[ ]:
```
# Load data from CSV files
X_train = np.loadtxt('X_train.csv', delimiter=',')
y_prediction = np.loadtxt('y_prediction.csv', delimiter=',')

# Find optimal i-parameter using 5-fold cross-validation
def i_param_selection(x, y):
    # Testing i values {0, 99}
    i_values = range(0, 100)

    # Variables to store the (currently) best i-parameter and its C-index
    best_i = None
    best_c_index = -9999

    for i in i_values:
        # Store the C-index for 5 folds
        c_index_arr = []

        kf5 = KFold(n_splits=5, shuffle=True, random_state=7)
        # Provides 5 folds of train/test indices
        for train_index, test_index in kf5.split(x):
            # Split the data into train/test according to the fold
            x_train, x_test = x[train_index], x[test_index]
            y_train, y_test = y[train_index], y[test_index]

            # Calculate Y prediction with use_ith_feature and
            # then calculate C-index with the predictions
            y_pred_val = use_ith_feature(x_train, y_train, x_test, i)
            c_index_val = cindex(y_test, y_pred_val)
            c_index_arr.append(c_index_val)

        # Calculate the mean C-index for single i-parameter
        c_indexes_np = np.array(c_index_arr)
        mean_c_index = np.mean(c_indexes_np)
        # Replace best i-parameter and C-index if better than current best
        if mean_c_index > best_c_index:
            best_c_index = mean_c_index
            best_i = i
```

```
    return best_i, best_c_index

best_i, five_fold_c_index = i_param_selection(X_train, y_prediction)
```

[ ]:
```
# Estimate the C-index using nested 5-fold cross-validation
def nested_c_estimation(x, y):

    # Store the C-index for 5 folds
    outer_c_arr = []

    kf5_outer = KFold(n_splits=5, shuffle=True, random_state=14)
    # Provides 5 folds of outer train/test indices, the train indices are used␣
 ↪for both training and validation
    for train_index, test_index in kf5_outer.split(x):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # We can use the i_param_selection function as the inner CV
        # The outer test indices are NOT used in feature selection
        inner_best_i = i_param_selection(x_train, y_train)[0]

        # Then we test the best i-parameter on the outer, yet unused test data
        y_pred_val = use_ith_feature(x_train, y_train, x_test, inner_best_i)
        outer_c_index = cindex(y_test, y_pred_val)
        outer_c_arr.append(outer_c_index)

    # Calculate and return the mean C-index from the outer folds
    outer_c_arr_np = np.array(outer_c_arr)
    mean_outer_c = np.mean(outer_c_arr_np)

    return mean_outer_c

n_5_fold_c_index = nested_c_estimation(X_train, y_prediction)
```

[ ]:
```
# Print the best i-parameter and its C-index
print(f"With regular CV the best i-parameter was: {best_i} with C-index of:␣
 ↪{five_fold_c_index}")
print(f"Using nested CV the average C-index was: {n_5_fold_c_index} which is␣
 ↪{five_fold_c_index-n_5_fold_c_index} lower than the C-index from regular CV")
```

```
With regular CV the best i-parameter was: 5 with C-index of: 0.74
Using nested CV the average C-index was: 0.5533333333333333 which is
0.18666666666666665 lower than the C-index from regular CV
```

The nested CV gives lower C-index average (0.55) compared to the non-nested best C-index value (0.74). Seems logical since the nested CV evaluates the performance using left-out test data, which better represents the models performance on new data.