exercise-4_lkmail

February 21, 2024

1 TKO_7092 Evaluation of Machine Learning Methods 2024

Student name: Lauri Maila

Student number: 2209361

Student email: lkmail@utu.fi

1.1 Exercise 4

Complete the tasks given to you in the letter below. In your submission, explain clearly, precisely, and comprehensively why the cross-validation described in the letter failed, how cross-validation should be performed in the given scenario and why your cross-validation will give a reliable estimate of the generalisation performance. Then implement the correct cross-validation for the scenario and report its results.

Remember to follow all the general exercise guidelines that are stated in Moodle. Full points (2p) will be given for a submission that demonstrates a deep understanding of cross-validation on pair-input data and implements the requested cross-validation correctly (incl. reporting the results). Partial points (1p) will be given if there are small error(s) but the overall approach is correct. No points will be given if there are significant error(s).

The deadline of this exercise is **Wednesday 21 February 2024 at 11:59 PM**. Please contact Juho Heimonen (juaheim@utu.fi) if you have any questions about this exercise.

Dear Data Scientist,

I have a long-term research project regarding a specific set of proteins. Currently I am attempting to discover small organic compounds that can bind strongly to these proteins and thus act as drugs. I have a list of over 100.000 potential drug molecules, but their affinities still need to be verified in the lab. Obviously I do not have the resources to measure all the possible drug-target pairs, so I need to prioritise. I have decided to do this with the use of machine learning, but I have encountered a problem.

Here is what I have done so far: First I trained a K-nearest neighbours regressor with the parameter value K=10 using all the 400 measurements I had made in the lab, which comprise of all the 77 target proteins of interest but only 59 different drug molecules. Then I performed a leave-one-out

cross-validation with this same data to estimate the generalisation performance of the model. I used C-index and got a stellar score above 90%. Finally I used the model to predict the affinities of the remaining drug molecules. The problem is: when I selected the highest predicted affinities and tried to verify them in the lab, I found that many of them are much lower in reality. My model clearly does not work despite the high cross-validation score.

Please explain why my estimation failed and how leave-one-out cross-validation should be performed to get a reliable estimate. Also, implement the correct leave-one-out cross-validation and report its results. I need to know whether I am wasting my lab resources by using my model.

The data I used to create my model is available in the files `input.data`, `output.data` and `pairs.data` for you to use. The first file contains the features of the pairs, whereas the second contains their affinities. The third file contains the identifiers of the drug and target molecules of which the pairs are composed. The files are paired, i.e. the i*th* row in each file is about the same pair.

Looking forward to hearing from you soon.

Yours sincerely,
Bio Scientist

Answer the questions about cross-validation on pair-input data

[ ]:
```
# Why did the estimation described in the letter fail?
# > Because data is pair-input observations, and the regular loocv is not␣
 ↪suitable for this kind of data.
# There are dependencies between the pairs of observations, which skews the␣
 ↪results to be overly optimistic.
#
# How should leave-one-out cross-validation be performed in the given scenario␣
 ↪and why?
# > The training observations will be chosen for different out-of-sample A, B,␣
 ↪C and D.
# A: No restriction on the training observations
# B: The target protein of test observation is not allowed to be in the␣
 ↪training observations
# C: The drug of test observation is not allowed to be in the training␣
 ↪observations
# D: Both target protein and drug of test observation are not allowed to be in␣
 ↪the training observations
# This will give 4 different C-index scores for each out-of-sample type
```

Import libraries

[ ]:
```
# Import the libraries you need.
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
```

Write utility functions

[ ]:
```
# I'm using the same C-index function that was provided in the first exercise
### Function for calculating C-index ###
# y: array containing true label values.
# yp: array containing the predicted label values.
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
```

Load datasets

[ ]:
```
# Read the data files (input.data, output.data, pairs.data).
df_input = pd.read_csv("input.data", delimiter=" ", header=None)
df_output = pd.read_csv("output.data", delimiter=" ", header=None)
df_pairs = pd.read_csv("pairs.data", delimiter=" ", header=None, names=['D',␣
 ↪'T'])
# Print shapes of data to check if they are read correctly
print("Input shape:",df_input.shape)
print("Output shape:",df_output.shape)
print("Pairs shape:",df_pairs.shape)
```

```
Input shape: (400, 67)
Output shape: (400, 1)
Pairs shape: (400, 2)
```

Implement and run cross-validation

[ ]:
```
# Implement and run the requested cross-validation. Report and interpret its␣
 ↪results.

### Function for special leave-one-out cross-validation (LOOCV) ###
def pair_input_cv(k, df_pairs, df_in, df_out):

    # We'll calculate C-index for all 4 different types of out-of-sample␣
 ↪observations
    types = ["A", "B", "C", "D"]
```

```
    # Number of input rows
    n = df_in.shape[0]

    # Store actual values and predictions for C-index calculation
    A_true_values = []
    A_predictions = []
    B_true_values = []
    B_predictions = []
    C_true_values = []
    C_predictions = []
    D_true_values = []
    D_predictions = []

    # Loop over each row to use as test set
    for i in range(n):
        for type in types:
            X = df_in
            y = df_out

            # Use the i-th data point as test set
            X_test = df_in.iloc[[i]]
            y_test = df_out.iloc[[i]]

            if type == "B":
                # The target protein of test observation is not allowed to be␣
↪in the training observations
                indices_to_remove = df_pairs.index[df_pairs['T'] ==␣
↪df_pairs['T'].iloc[i]].tolist()
            elif type == "C":
                # The drug of test observation is not allowed to be in the␣
↪training observations
                indices_to_remove = df_pairs.index[df_pairs['D'] ==␣
↪df_pairs['D'].iloc[i]].tolist()
            elif type == "D":
                # Both target protein and drug of test observation are not␣
↪allowed to be in the training observations
                indices_to_remove = df_pairs.index[(df_pairs['D'] ==␣
↪df_pairs['D'].iloc[i]) | (df_pairs['T'] == df_pairs['T'].iloc[i])].tolist()
            else:
                # A, no restriction on the training observations
                indices_to_remove = []

            # Add current test index to be removed
            indices_to_remove.append(i)
```

```
            # Drop the indices from the training data
            X_train = X.drop(index=indices_to_remove, inplace=False)
            y_train = y.drop(index=indices_to_remove, inplace=False)

            # Fit the kNN model
            knn = KNeighborsRegressor(n_neighbors=k)
            knn.fit(X_train, y_train)

            # Predict model on the left out data point
            y_pred = knn.predict(X_test)

            # Store the predictions and true values
            if type == "A":
                A_predictions.append(y_pred)
                A_true_values.append(y_test.values.flatten())
            elif type == "B":
                B_predictions.append(y_pred)
                B_true_values.append(y_test.values.flatten())
            elif type == "C":
                C_predictions.append(y_pred)
                C_true_values.append(y_test.values.flatten())
            elif type == "D":
                D_predictions.append(y_pred)
                D_true_values.append(y_test.values.flatten())


    # At the end, calculate the C-index using provided function
    cind_A = cindex(A_true_values, A_predictions)
    cind_B = cindex(B_true_values, B_predictions)
    cind_C = cindex(C_true_values, C_predictions)
    cind_D = cindex(D_true_values, D_predictions)

    return [cind_A, cind_B, cind_C, cind_D]


# Using k=10 for kNN as stated in case
k = 10

# Calculate C-index for each case A, B, C and D
c_indexes = pair_input_cv(k, df_pairs, df_input, df_output)
print(f"C-indexes: A: {round(c_indexes[0],3)} B: {round(c_indexes[1],3)} C:␣
 ↪{round(c_indexes[2],3)} D: {round(c_indexes[3],3)}")

# As a result we get C-indexes: A: 0.83 B: 0.83 C: 0.52 D: 0.522
# This time we get more realistic results, where performance in cases C and D␣
 ↪are significantly lower
```

```
# as expected since only some of the drugs molecules were used in the␣
 ↪observations, whereas all the target proteins were used.
```

```
C-indexes: A: 0.83 B: 0.83 C: 0.52 D: 0.522
```