## ECE475 Project 3 - Nishat Ahmed, Seyun Kim & Lucia Rhode

Re-implement the example in section 7.10.2 using any simple, out of the box classifier (like K nearest neighbors from sci-kit). Reproduce the results for the incorrect and correct way of doing cross-validation.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from collections import Counter


def most_find(sequence, n):
    lst = sorted(range(len(sequence)), key=lambda x:sequence[x], reverse=True)
    return lst[:n]


def percent_correct(y_hat, y):
  return sum(y_hat == y) / len(y)


# create fake predictors
x = np.zeros((50, 5000))
for i in range(50):
  x[i,:] = np.random.normal(0, 1, 5000)
print(x)
```

```
[[ 0.88521285 -1.24008161  1.15350889 ...  1.64383643 -0.81465598
   0.48432619]
 [ 1.93658042 -0.57339435  1.17527517 ...  1.20784758 -1.38952118
  -1.82605635]
 [ 1.36264709  0.1061386  -0.14932739 ... -0.238464    1.88703506
   1.38234044]
 ...
 [-1.15016658 -2.28548511  2.3357725  ... -1.163781   -0.12009878
  -1.14638564]
 [-1.52242922  0.7315752  -0.45052919 ... -0.17544105 -1.80644355
   1.13810271]
 [-0.87020905  1.28829754 -1.4792499  ...  1.09649444  0.4797836
   0.97655453]]
```

```
# create fake labels
y = np.random.randint(2, size=50)
```

1. Done in the wrong way

```
# screen the predictors
corr = np.corrcoef(x, y, rowvar=False)
correlation = corr[:,-1][:-1]

# find the indices of the 100 most significant predictors
corr_ind = most_find(correlation, 100)

x_predictors = x[:, corr_ind]



# perform k-fold cross validation
scores = []
kf = KFold(n_splits=50)

for train_index, test_index in kf.split(x_predictors):
  x_train, x_test = x_predictors[train_index], x_predictors[test_index]
  y_train, y_test = y[train_index], y[test_index]

  neigh = KNeighborsClassifier(n_neighbors=1)
  neigh.fit(x_train, y_train)

  y_hat = neigh.predict(x_test)
  score = percent_correct(y_hat, y_test)
  scores.append(score)



# report % correct
print(f'percent correct: {np.mean(scores) * 100}%')
print(f'error rate: {(1-np.mean(scores))*100}%')
```

```
percent correct: 98.0%
error rate: 2.000000000000018%
```

2. Correct Way

```
scores = []
kf = KFold(n_splits=50)

for train_index, test_index in kf.split(x):
  corr = np.corrcoef(x[train_index], y[train_index], rowvar=False)
  correlation = corr[:,-1][:-1]

  # find the indices of the 100 most significant predictors
  corr_ind = most_find(correlation, 100)
  x_predictors = x[:, corr_ind]
  x_train, x_test = x_predictors[train_index], x_predictors[test_index]
  y_train, y_test = y[train_index], y[test_index]

  neigh = KNeighborsClassifier(n_neighbors=1)
```

```
  neigh.fit(x_train, y_train)

  y_hat = neigh.predict(x_test)
  score = percent_correct(y_hat, y_test)
  scores.append(score)


# report % correct
print(f'percent correct: {np.mean(scores) * 100}%')
print(f'error rate: {(1-np.mean(scores))*100}%')
```

```
percent correct: 50.0%
error rate: 50.0%
```

The error rate for the incorrect way was much lower than for the correct way of doing cross-validation. This is because in the incorrect way, the predictors are chosen on the basis of all the samples meaning it has seen the test set.