K-nearest neighbors and Cross Validation

Team assignment: You are only allowed to discuss the specifics of this project with your teammates and the instructor for this assignment.

Purpose

The purpose of this assignment is to explore a typical machine learning pipeline to evaluate and tune a classification model.

Task

Download the starter code and the red wine data set.

Note: While there are automated solutions to many of these tasks, you will learn more by implementing them. The knowledge you gain from manually constructing these workflows will help you when you need to integrate with other packages or control certain aspects to the execution or try novel variations.

Restricted tools: The following tools are worth knowing about, but do not use them for this assignment. We want you to learn first-hand to do these tasks due to their importance. Do not import any members from: **sklearn.model_selection.**

Complete the following tasks.

Part 1: Tools for cross validation and out-of-sample performance estimation

For this part of the project, you will develop some helper functions to prepare datasets for cross validation.

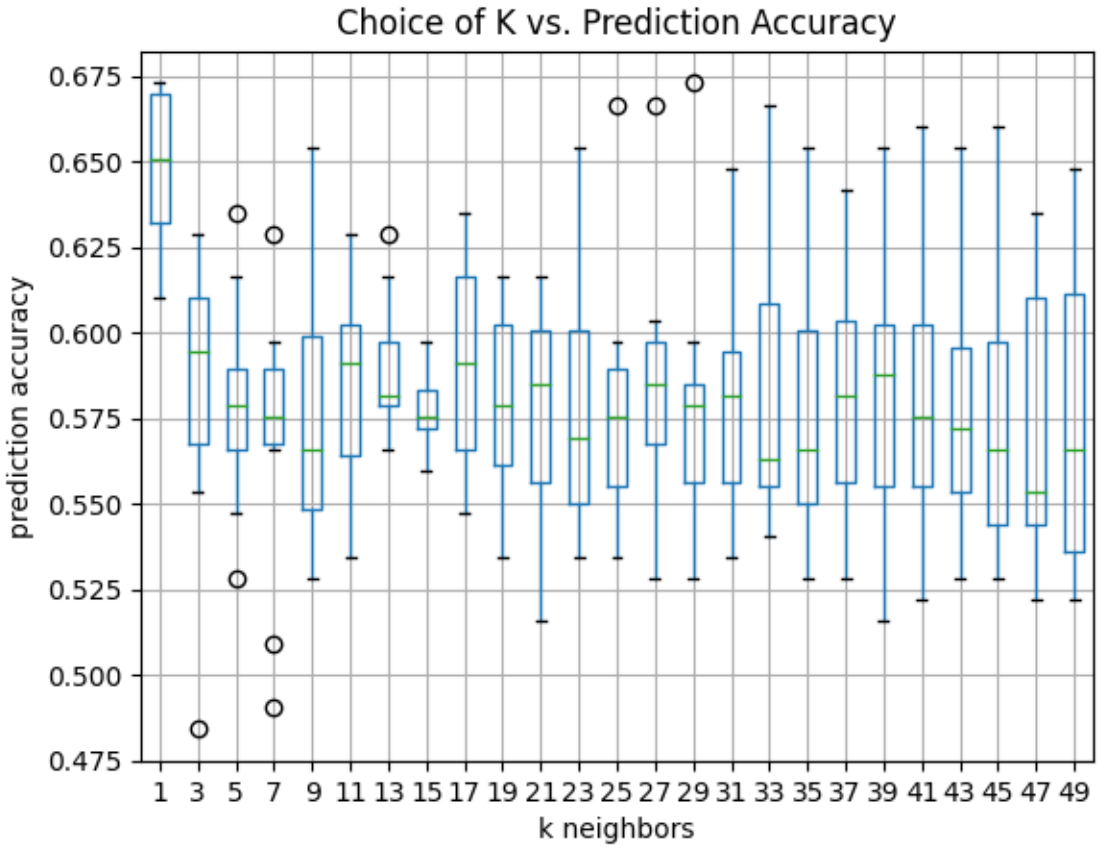
1. Create the function: `create_fold_indexes(df, n_folds = 10)`
    a. This function should take a pd.DataFrame and a number of folds as input parameters.
    b. Have the function create a list of row indexes in random order.
    c. Calculate a set size for each fold. This would be the number of the rows to place in each fold.
        i. Divide the number or rows by the number of folds to get the “chunk” / fold size.
        ii. Remember to round the fold size to an integer.
    d. Insert into separate lists a number of indexes equal to the estimated fold size.
        i. The last fold may have fewer than the rest due to rounding.
    e. Your function should return a list of lists (which could be a dataframe or numpy ndarray) with each element a list of indexes for the specific fold.
        i. This means fold indexes could look like [[13,46,312, 119 …], [145, 317, 18, …], …]
        ii. Return this list of lists.
2. Create a function: `train_test_index_split(fold_indexes, current_test_index)`
    a. This function will take the fold indexes and return two lists.
    b. The two lists will be the training example index list and the test example index list.
    c. Using the current_test_index, make that index set the test indexes.
    d. For the remaining index lists, combine them into a single list to return as the training index list.

e. Use Python's tuple return to return both training_indexes and test_indexes in the same statement, e.g. return train_indexes, test_indexes.

Part 2: Experiment with K-NN.

In this part of the project, you will explore the performance of a k-nearest neighbor classification algorithm for different choices of k on the red wine data set.

1. In main, load the red wine dataset into a pd.DataFrame.
2. Use your create_fold_indexes function to generate 10 folds for the red wine dataset.
    a. Set your random see to 42 using np.random.seed(42)
3. Evaluate a single classifier using cross validation on your 10 folds.
    a. This will not be in your final source code, but it represents part of the inner loop you will need in the final submission.
    b. Cycle through each of the 10 folds and do the following:
        i. Call the train_test_index_split function to get training and test indexes.
        ii. From these indexes, create two data sets, one for training and one for testing.
        iii. Standardize your training data and construct your K-NN classifier using the standardized data.
            1. Use StandardScaler from sklearn.preprocessing
            2. Use KNeighborsClassifier from sklearn.neighbors
        iv. Standardize your test data using the StandardScaler from your training set.
        v. Make your predictions on the test set and evaluate the prediction accuracy. Use accuracy_score from sklearn.metrics for this task.
        vi. As you calculate the accuracy for each fold, keep track of each test-performance estimate. The simplest way would be to append them to a list.
    c. Once you have completed a full cycle through all folds, print the mean accuracy and standard deviation of the accuracy for the classifier.
4. After completing task 2.3, Modify your code to search through different values for k.
    a. Loop through the values 1 to 49 for k, increasing by 2 at each step. This gives a sequence of k=1, 3, 5 … to k=49. Use 50 as the end for range.
    b. For each k, repeat the steps from 2.3 to estimate the out-of-sample performance accuracy of the algorithm.
    c. After the fold accuracy has been calculated, print the mean accuracy and standard deviation of the accuracy.
    d. Save the accuracy estimates for the 10 folds to a table.
    e. After you have completed the run for all k values, plot a boxplot of the accuracy estimates. You should get a figure that looks like the one below.



5. Repeat the process from task 4 with standardizing scaling omitted.
    a. Do not scale the training set or the test set and repeat your estimates of the out-of-sample prediction accuracy.
    b. Create another plot of the k values vs. the prediction accuracy.
    c. Give a one paragraph description of the effect the standardized scaling makes on the data.
6. In a Word document, copy your two plots into the document and give the paragraph from task 5 at the end.
    a. Here you should explain the difference between the two plots and give your summary of the effect of standardizing scaling on this data set.
    b. Finally answer the question, “Which classifier (including processing steps) would you choose to deplore for this classification task base on this evidence?”

Submission

Submit your source code and Word document to Moodle as a zipped file. Make sure both source code and the Word document contain the names of all students.

Evaluation

You will be evaluated based on the following criteria.

/2 Header comments with member names

/3 Program runs without errors and need no modification

/5 `create_fold_indexes` correctly returns randomized index sets.

/5 `train_test_index_split` Correctly produces the training and test indexes

/10 Values of K explored using correct classifiers, plot is correct

/5 Values of K are explored without using standardize scaling

/5 Written explanation directly addresses scaling and performance.

/35 Total Points

Example Output

```
k= 1 mean accuracy =  0.6484276729559748 std = 0.022471995297042534
k= 3 mean accuracy =  0.5830188679245283 std = 0.042143058032077865
k= 5 mean accuracy =  0.5792452830188679 std = 0.03115871069767585
k= 7 mean accuracy =  0.5691823899371069 std = 0.040786323812041336
k= 9 mean accuracy =  0.5748427672955974 std = 0.04034754456593512
k= 11 mean accuracy =  0.5855345911949685 std = 0.029267628952270447
k= 13 mean accuracy =  0.5899371069182389 std = 0.02006454694172813
k= 15 mean accuracy =  0.5773584905660377 std = 0.011405845290344184
k= 17 mean accuracy =  0.5911949685534591 std = 0.03301470806713547
k= 19 mean accuracy =  0.5779874213836479 std = 0.029566438424007937
k= 21 mean accuracy =  0.5748427672955976 std = 0.035230107400587005
k= 23 mean accuracy =  0.5779874213836477 std = 0.03882087889814078
k= 25 mean accuracy =  0.5792452830188679 std = 0.03636576390231527
k= 27 mean accuracy =  0.5849056603773585 std = 0.03726706254639659
k= 29 mean accuracy =  0.5792452830188679 std = 0.0390466492162514
k= 31 mean accuracy =  0.5811320754716981 std = 0.032504901372434884
k= 33 mean accuracy =  0.5817610062893082 std = 0.04078632381204133
k= 35 mean accuracy =  0.5792452830188679 std = 0.03982676779619512
k= 37 mean accuracy =  0.5830188679245284 std = 0.04130032244333677
k= 39 mean accuracy =  0.580503144654088 std = 0.04044002877783461
k= 41 mean accuracy =  0.5817610062893082 std = 0.04067842329545545
k= 43 mean accuracy =  0.5779874213836478 std = 0.03802012651674181
k= 45 mean accuracy =  0.5779874213836478 std = 0.042907854685228855
k= 47 mean accuracy =  0.5729559748427672 std = 0.043112227619105804
k= 49 mean accuracy =  0.5754716981132075 std = 0.04605006842031193
k= 1 mean accuracy =  0.6044025157232704 std = 0.048030809626737225
k= 3 mean accuracy =  0.5144654088050313 std = 0.03652855599937022
k= 5 mean accuracy =  0.5069182389937108 std = 0.04349283850696834
k= 7 mean accuracy =  0.49371069182389943 std = 0.04518293933386227
k= 9 mean accuracy =  0.49685534591194963 std = 0.05118052669070734
k= 11 mean accuracy =  0.49748427672955975 std = 0.04558965209005781
```

```
k= 13 mean accuracy =  0.5081761006289308 std = 0.03676840464300839
k= 15 mean accuracy =  0.5050314465408805 std = 0.03631738907025616
k= 17 mean accuracy =  0.5075471698113206 std = 0.03496086065330137
k= 19 mean accuracy =  0.5113207547169811 std = 0.04172381798336934
k= 21 mean accuracy =  0.5232704402515723 std = 0.045526946090198375
k= 23 mean accuracy =  0.5132075471698114 std = 0.05068001381431217
k= 25 mean accuracy =  0.5226415094339624 std = 0.03938287800435993
k= 27 mean accuracy =  0.5276729559748427 std = 0.04490975127539067
k= 29 mean accuracy =  0.5289308176100629 std = 0.03280771726131982
k= 31 mean accuracy =  0.5320754716981131 std = 0.043391668706063743
k= 33 mean accuracy =  0.5257861635220126 std = 0.02997243773695851
k= 35 mean accuracy =  0.5238993710691823 std = 0.04033120180772766
k= 37 mean accuracy =  0.5188679245283019 std = 0.039804690872217025
k= 39 mean accuracy =  0.5257861635220126 std = 0.0436944753714483
k= 41 mean accuracy =  0.5232704402515724 std = 0.03886048339030107
k= 43 mean accuracy =  0.5264150943396226 std = 0.048721282253783034
k= 45 mean accuracy =  0.5188679245283019 std = 0.03799699991696406
k= 47 mean accuracy =  0.5157232704402517 std = 0.04255300446579546
k= 49 mean accuracy =  0.5169811320754717 std = 0.0443533737244709
```