Attila Koksal

862058705

3/18/2025

CS170: Project 2: Feature Selection Nearest Neighbor

All code is original, except:

- Iris dataset - https://archive.ics.uci.edu/dataset/53/iris
- Lecture slides from machine learning

Introduction

The nearest neighbor technique must be used to implement feature selection in this project. After that is finished, I need to put forward selection and backward selection into practice, which are two greedy search algorithms. I tested my method on other datasets with different feature counts. I tested my code using three datasets with 10, 20, and 80 features—which are available in the folder mentioned in the project specification PDF—as well as one real-world dataset with four features. Finding the best characteristics for classification is the aim of these tests.

Implementation

User Interface

I created an interface that is easy to use. The filename, which must be in the same folder as the code, is first requested to be entered by the user. They must press 1 or 2 to select the forward or backward selection algorithms after entering the filename. After that, the user is prompted to enable sampling. They can choose "No" to turn off sampling, or "Yes," which will require them to enter the sample rate.

Implementation Details

I decided to use Python to build the algorithms and made use of two libraries: the itertools library's combinations function to create feature combinations with ease, and NumPy for effective array manipulations. Several utility functions are included in our code, including a function to construct combinations based on length using list(combinations(list, length)) and a Euclidean distance function built using np.linalg.norm(). The file reader function, which loads datasets from a specified file, is another essential tool. Real-world and synthetic datasets are handled differently by distinct reader functions (or if-else expressions). I chose the IRIS dataset for the real-world dataset, established the classification target at the start, and normalized the numerical values. I transform the text from IEEE format to floating-point values after first separating the text into discrete floating-point integers for synthetic datasets.

Nearest Neighbor

Three arguments are required for the nearest neighbor algorithm function to produce an integer array with the distances between the test sample and the samples in the input dataset. Using a for loop, it iterates over each sample, determining whether the sample index deviates from the test sample index. It determines the Euclidean distance between the samples if they differ. Otherwise, in order to keep the test sample from being regarded as the nearest neighbor, the distance between it and itself is set to an extremely big integer value.

Forward Selection

The forward selection method begins by determining each feature's correctness, choosing the best one, and then deleting it from the feature list. The selected feature is appended to the output list following each iteration. Next, I use get_combinations_with_length() to create combinations of the chosen feature with each of the other features one at a time. Next, I determine the optimal combination by calculating the accuracy for each pair. This procedure keeps going until every feature has been assessed, at which point the combination with the best accuracy is identified. Because one feature with the highest accuracy is eliminated in each iteration, this method employs a greedy elimination technique.

Backward Elimination

The backward elimination process starts with the entire collection of features, in contrast to the forward selection algorithm. Combinations with N−1 features—where N is the total number of features—are evaluated in the first iteration. The missing characteristic is detected for removal, and the combination with the highest accuracy is chosen. With the exception of those that have previously been removed, the combinations in the subsequent iteration comprise N−2 features. Until the ideal feature subset is identified, this process is repeated, removing one characteristic at a time depending on the highest-accuracy combination.

Sampling Rate

The time needed for feature selection rises with the size of the dataset. In this project, I used a sampling technique to maintain the code's efficiency. I only predict labels for a subset of samples, as defined by dividing the entire sample count by the sampling rate, rather than predicting labels for every sample and calculating accuracy. The most pertinent characteristics are then found by applying the feature selection function to this subset. Lastly, the complete dataset is used to compute accuracy. This method facilitates precise evaluation and effective feature selection by reducing computing difficulties in huge datasets.

Results

Dataset	Algorithm	Sampling Rate	Best Accuracy	Best Features	Execution Time
Small Dataset	Forward	None	96.00%	[3, 4]	1.14 secs
	Backward	None	96.00%	[3, 4]	1.18 secs
	Forward	15	95.33%	[1, 2, 3, 4]	0.33 secs
	Backward	15	95.33%	[1, 2, 3, 4]	0.39 secs
Large Dataset	Forward	10	95.33%	[1, 2, 3, 4]	0.19 secs
	Backward	10	95.33%	[1, 2, 3, 4]	0.26 secs
	Forward	50	95.33%	[1, 2, 3, 4]	0.43 secs
	Backward	50	95.33%	[1, 2, 3, 4]	0.66 secs
Real-World	Forward	None	96.00%	[3, 4]	1.14 secs
Dataset: IRIS	Backward	None	96.00%	[3, 4]	1.86 secs
	Forward	2	95.33%	[1, 2, 3, 4]	0.68 secs

	Backward	2	95.33%	[1, 2, 3, 4]	0.73 secs

Table 1: Execution Results

As anticipated, Table 1 illustrates that the outcomes of forward selection and backward elimination are almost the same. Both algorithms attain the utmost accuracy and find the same ideal characteristics. Their execution time is the primary distinction. While backward elimination starts by eliminating the lowest-accuracy features, which makes its process relatively longer, forward selection is typically faster since it gradually chooses the highest-accuracy characteristics.

I employ sampling for large datasets because it offers the best accuracy rate at a much shorter execution time. Although speed and accuracy must be traded off, the outcomes are still very effective for big datasets. The method first finds irrelevant features and spends too much time on them, which prevents it from accessing higher-accuracy features within the specified sampling rate. This is the reason for the decreased accuracy in the Iris dataset when backward elimination with sampling is used. Furthermore, both forward selection and backward elimination make greedy choices that, in reality, may produce distinct results.

I have chosen the small and large datasets at random from the folder provided named P2_datasets like the dataseys, CS170_Small_Data__15.txt and CS170_Large_Data__99.txt. The real-world dataset I used was called the IRIS dataset, which I got from the UCI Machine Learning Repository. The dataset has the following characteristics:

- Instances: 150
- Attributes: 4
- Sepal length (cm): continuous
- Sepal width (cm): continuous
- Petal length (cm): continuous

- Petal width (cm): continuous
- Class: string:
  - Iris Setosa
  - Iris Versicolour
  - Iris Virginica

There are three classes in the Iris dataset; two of the classes are not linearly separable from one another, whereas the third class is. Furthermore, there is no initial normalization of the data between 0 and 1. To guarantee consistency in the dataset, I employ normalization during the data reading process.

Conclusion

To find the correct features, I learned how to use the nearest neighbor method for feature selection in this project. I also learned how to use forward selection and backward elimination. I discovered through my tests that forward selection and backward removal produced essentially the same outcomes. They did, however, occasionally diverge because of their greedy, decision-making processes.

Appendix A

Code Test Sample

Welcome to the Feature Selection algorithm

Type name of the file to test: CS170_Small_Data__15.txt

Type the name of algorithm you want to run:

1) Forward Selection

2) Backward Elimination

1

Do you want to apply sampling algorithm to reduce the time (Enter 1 or 2):

1) Yes

2) No

1

Enter your desired sampling rate (Enter an integer number):

10

This dataset has 4 features (not including the class attribute), with 150 instances

Beginning search:

Accuracy for features [1] = 100.0%

Accuracy for features [2] = 73.33%

Accuracy for features [3] = 86.67%

Accuracy for features [4] = 86.67%

Accuracy for features [1, 2] = 86.67%

Accuracy for features [1, 3] = 100.0%

Accuracy for features [1, 4] = 100.0%

Accuracy for features [1, 3, 2] = 80.0%

Accuracy for features [1, 3, 4] = 93.33%

Accuracy for features [1, 3, 4, 2] = 100.0%

The best accuracy achieved by features [1, 3, 4, 2] and accuracy = 95.33%

Elapsed time for this run: 0.41482019424438477 seconds

Code

FeatureSelectionNN.py:

```
import numpy as np

from itertools import combinations

import random

import time
```

```
datasetName = ['CS170_Small_Data__15.txt', 'CS170_Large_Data__18.txt',
'CS170_Large_Data__99.txt', 'iris.data']
nFeatures = [10, 20, 80, 4]


# User Interface Implementation
print('Welcome to the Feature Selection algorithm')
nameInput = input('Type name of the file to test: ')
print('\n')



if('small' in nameInput):
    index_dataset = 0
elif('XXXlarge' in nameInput):
    index_dataset = 2
elif('large' in nameInput):
    index_dataset = 1
else:
    index_dataset = 3



algorithmNum = input('Type the name of algorithm you want to run: \n1) Forward Selection\n2)
Backward Elimination\n')
print('\n')
```

```
samplingAlgorithm = int(input('Do you want to apply sampling algorithm to reduce the time
(Enter 1 or 2): \n1) Yes\n2) No\n'))
print('\n')

if samplingAlgorithm == 1:
    sampling_rate = int(input('Enter your desired sampling rate (Enter an integer number): \n'))
else:
    sampling_rate = 1



# Reading dataset
# Real world dataset
if (index_dataset == 3):
    with open(datasetName[index_dataset], "r") as file:
        file_content = file.readlines()
    for i in range(len(file_content)):
        file_content[i] = file_content[i][0:-1]
    file_content.remove('')

    # Get number of sanples
    num_samples = len(file_content)
```

```
# Process data
data = []

# Split columns
for i in range(num_samples):
    data.append(file_content[i].split(',')[0:5])

# Encode target classes to integer values
Encode_classes = [('Iris-setosa', 1), ('Iris-versicolor', 2), ('Iris-virginica', 3)]
for i in range(num_samples):
    if (data[i][4] == 'Iris-setosa'):
        data[i][4] = 1
    elif (data[i][4] == 'Iris-versicolor'):
        data[i][4] = 2
    else:
        data[i][4] = 3
    # Convert strings to integers
    data[i][0] = float(data[i][0])
    data[i][1] = float(data[i][1])
    data[i][2] = float(data[i][2])
    data[i][3] = float(data[i][3])
```

```
    # Convert list data to numpy array, and put target class to first column to math dataset to
previous format
    data = np.array(data)
    data = data[:,[4,0,1,2,3]]

    # Normalize the data using (0-1) normalization
    min_values = np.min(data, axis=0)
    max_values = np.max(data, axis=0)
    data = (data - min_values) / (max_values - min_values)

    print(f'This dataset has {nFeatures[index_dataset]} features (not including the class attribute),
with {num_samples} instances\n')

# Synthetic dataset
else:
    # Open file in binary mode
    with open(datasetName[index_dataset], 'rb') as file:
        ascii_text = file.read().decode('ascii') # Read the ASCII text data

    # Split the text into individual floating-point numbers:
    numbers = ascii_text.split()

    num_samples = int(len(numbers)/(nFeatures[index_dataset]+1))
```

```
    # convert the IEEE format to floating point numbers
    data = []
    for num in numbers:
        if (num[0] == '-'):
            data.append(float(num[0:10]) * 10**(int(num[11:])))
        else:
            data.append(float(num[0:9]) * 10**(int(num[10:])))
    data = np.array(data).reshape((num_samples, nFeatures[index_dataset]+1)) # This is the data
we can work on it

    print(f'This dataset has {nFeatures[index_dataset]} features (not including the class attribute),
with {num_samples} instances\n')

# Forward selection for feature selection
def getCombinationsWithLen(lst, length):
    return list(combinations(lst, length))

def forwardSelection(data, nFeatures, samplingAlgorithm, samp_rate = 100):
    num_test_org, _ = data.shape
    print('Beginning search:\n')
    best_feat_so_far = []
    best_feat = []
```

```
best_acc = 0.0
temp = []

if samplingAlgorithm == 1:
    num_test = int(num_test_org/samp_rate)
else:
    num_test = num_test_org

for L in range(1, nFeatures+1):
    temp  = list(range(1, nFeatures+1))
    if len(best_feat_so_far) != 0:
        for item in best_feat_so_far:
            temp.remove(item)
    else:
        temp = list(range(1, nFeatures+1))
    combinations_list = getCombinationsWithLen(temp, L-len(best_feat_so_far))
    acc = 0.0
    best_so = []
    for item in list(combinations_list):
        feat_ind = best_feat_so_far + list(item)
        label_pred = []
        ind_range = random.sample(range(0, num_test_org), num_test)
        for i in ind_range:
```

```
            label_pred.append(nearestNeighbor(data[:,[0]+feat_ind], data[i,feat_ind], i))
        correct = 0
        k = 0
        for i in ind_range:
            if (int(data[i,0]) == label_pred[k]):
                correct += 1
            k += 1
        print(f'Accuracy for features {feat_ind} = {round(correct/num_test*100, 2)}%')
        if ( (correct/num_test*100) > acc):
            acc = (correct/num_test*100)
            best_so = feat_ind
            if(acc >= best_acc):
                best_acc = acc
                best_feat.append(feat_ind)
    best_feat_so_far = best_so
    print('\n')

acc_list = []
for  feat_ind in best_feat:

    label_pred = []
    for i in range(num_test_org):
        label_pred.append(nearestNeighbor(data[:,[0]+feat_ind], data[i,feat_ind], i))
```

```
        correct = 0

        for i in range(num_test_org):
            if (int(data[i,0]) == label_pred[i]):
                correct += 1

        acc_list.append(correct/num_test_org*100)

    max_ind = acc_list.index(max(acc_list))

    print(f'\n\nThe best accuracy achieved by features {best_feat[max_ind]} and accuracy =
{round(acc_list[max_ind], 2)}%\n\n')


# Nearest Neighbor function

def distFunction(x, y):
    dist = np.linalg.norm(x-y)
    return dist

def nearestNeighbor(dataset, test_sample, index_data):
    num_samples, _ = dataset.shape
    Dist_list = []
```

```
    for i in range(num_samples):
        if (i != index_data):
            Dist_list.append(distFunction(dataset[i,1:], test_sample))
        else:
            Dist_list.append(10000000000.0)
    label = int(dataset[Dist_list.index(min(Dist_list)), 0])
    return label

# Backward Elimination for feature selection
def getCombinationsWithLen(lst, length):
    return list(combinations(lst, length))


def backwardElimination(data, nFeatures, samplingAlgorithm, samp_rate = 100):
    num_test_org, _ = data.shape
    print('Beginning search:\n')
    worst_feat_so_far = []
    best_feat = []
    best_acc = 0.0
    temp = list(range(1, nFeatures+1))

    if samplingAlgorithm == 1:
        num_test = int(num_test_org/samp_rate)
    else:
```

```
    num_test = num_test_org


for L in range(0, nFeatures):
    # Add availabels:
    temp  = list(range(1, nFeatures+1))
    if len(worst_feat_so_far) != 0:
        for item in worst_feat_so_far:
            temp.remove(item)
    else:
        temp = list(range(1, nFeatures+1))
    combinations_list = getCombinationsWithLen(temp, nFeatures-L)
    acc = 0.0
    worst_so = []
    for item in list(combinations_list):
        feat_ind = list(item)
        label_pred = []
        ind_range = random.sample(range(0, num_test_org), num_test)
        for i in ind_range:
            label_pred.append(nearestNeighbor(data[:,[0]+feat_ind], data[i,feat_ind], i))
        correct = 0
        k = 0
        for i in ind_range:
            if (int(data[i,0]) == label_pred[k]):
```

```
            correct += 1
          k += 1
        print(f'Accuracy for features {feat_ind} = {round(correct/num_test*100, 2)}%')

        if ( (correct/num_test*100) > acc):
          acc = (correct/num_test*100)
          worst_so = list(range(1, nFeatures+1))
          if len(feat_ind) != 0:
            for item in feat_ind:
              worst_so.remove(item)
          if(acc >= best_acc):
            best_acc = acc
            best_feat.append(feat_ind)
      worst_feat_so_far = worst_so
      print('\n')

acc_list = []
for  feat_ind in best_feat:
    label_pred = []
    for i in range(num_test_org):
        label_pred.append(nearestNeighbor(data[:,[0]+feat_ind], data[i,feat_ind], i))
    correct = 0
```

```
        for i in range(num_test_org):
            if (int(data[i,0]) == label_pred[i]):
                correct += 1
        acc_list.append(correct/num_test_org*100)

    max_ind = acc_list.index(max(acc_list))

    print(f'\n\nThe best accuracy achieved by features {best_feat[max_ind]} and accuracy = 
{round(acc_list[max_ind], 2)}%\n\n')



# Run Feature selection algorithm on dataset
current_time = time.time()

if (algorithmNum == '1'):
    forwardSelection(data, nFeatures[index_dataset], samplingAlgorithm, sampling_rate)
else:
    backwardElimination(data, nFeatures[index_dataset], samplingAlgorithm, sampling_rate)

elapsed_time = time.time() - current_time

print("Elapsed time for this run: ", elapsed_time, "seconds\n\n")
```