Project 2: Feature Selection with Nearest Neighbor

Student Name1: Timofey Malko SID: 862311452 Lecture Session: 001

Solution:

CS170_Spring_2023_Small_data__53.txt	CS170_Spring_2023_Large_data__53.txt

Dataset	Best Feature Set	Accuracy
Small_data__53	Forward Selection = [2, 5, 7, 9]	0.87
Small_data__53	Backward Elimination = [4, 7]	0.85
Large_data__53	Forward Selection = [9, 19]	0.967
Large_data__53	Backward Elimination = [9, 19]	0.967

In completing this project, I consulted following resources:
Elearn with class lectures.
Clap - library for command line arguments parsing.
Rand - library for random number generation.
Itertools - library for more functions on iterators.
Contribution of each student in the group:
Tim - did project 2

Introduction

The program is modular with bearable performance, but not optimized(a lot of repeated distance calculations). Only has backward and forwards searches. Supports K-NN classifiers and can do K fold validation. Features are numbered starting with 0.

Challenges

Wanted to use a linear algebra library, but in the end failed to succeed.
Mistook min for max and spent days on one letter bug.

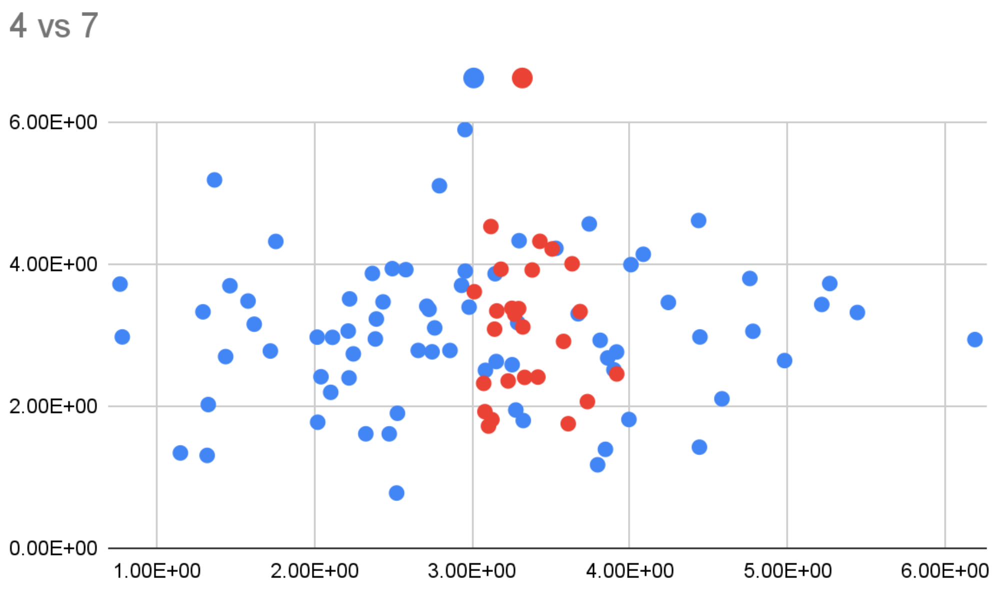
Code Design

There is a readme file with details on how to build and run the app.
There is a trait for a classifier with methods specified by project req and an instance for NNclassifier and an instance of KNNclassifier.
There is a trace for an evaluator and an instance for LooEvaluator. There is a method for validating by instance and validating by iterating over all instances with k folds (regular validator when k = 1).
The searches are interchangeable functions that take evaluator, classifier, and data.
Making the code modular.
Instances with features are stored in an InstanceArena, while active features are represented through an array of bools
The data from the file is normalized only if the -n flag is set.
not optimal but works.

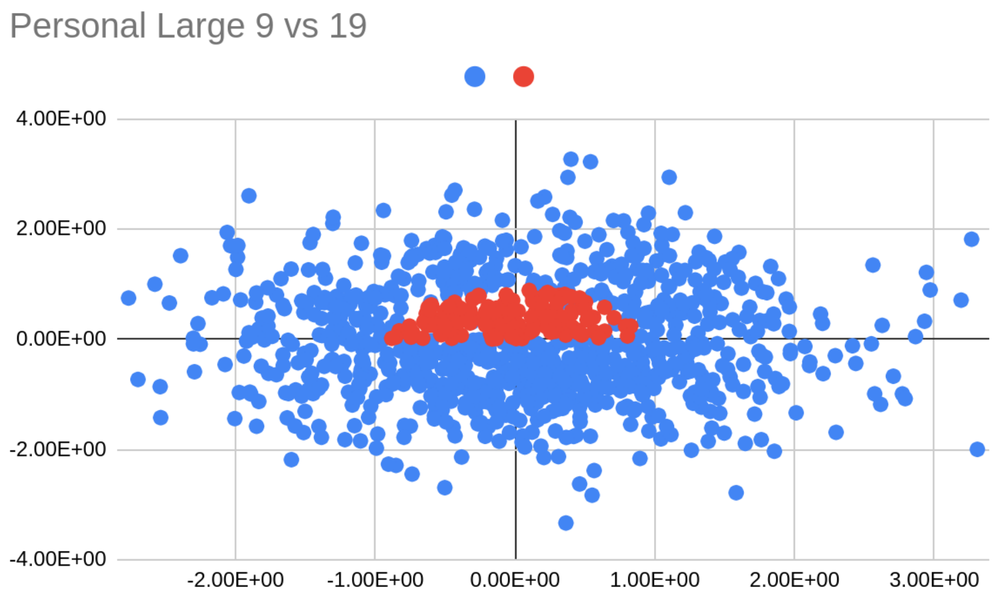
Dataset details

The General Small Dataset: Number of features 10, number of
Instances 100
The General Large Dataset: Number of features 40, number of
Instances 1000
Your Personal Small Dataset: Number of features 10, number of
Instances 100
Your Personal Large Dataset: Number of features 40, number of
Instances 1000

Small personal dataset
In 4v7 reds(1.0) form a cluster in the middle, with blue(2.0) spread all over.



Large personal dataset
In 9vs19 reds(1.0) form a cluster in the middles.



Algorithms

Forward Selection by testing a best unused feature to add, then adding it and repeating it until there are no more unused features with solution being the best iteration.

Backward Elimination by testing a worst used feature to remove, then removing it and repeating it until there are no more used features with solution being the best iteration.

Analysis

Forward Selection vs Backward Elimination Ex1

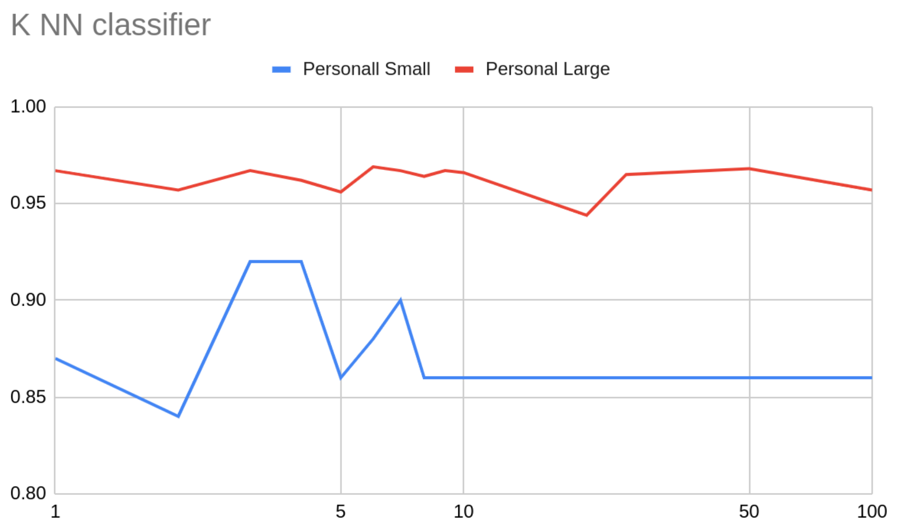
On a large dataset both forward and backward searches result in the same feature set and accuracy. Both reduce the feature set to the same two features. On a small dataset b. and f. are sharing only one feature￼ with different amounts of features in sets, but the accuracy is approximately the same. In both datasets the no features selection results in 72% accuracy, the large dataset benefits more from feature selection. Backwards elim. Is better when there is a relationship between features but it is slower. Forward elimination is better when the useful features set is expected to be not large. Overall both searches provide very similar results on these datasets.

Normalized vs not Normalized Ex2

Large datasets not normalized provide the same feature set with the same accuracy.
Small personal dataset not normalized forwards and backwards searches both dominated by feature [2] with 85% accuracy. This actually not bad compared to normalized version

	search	Small	Large	Personal Small	Personal Large
normalized	backward	[2, 4], 92%	[9, 19], 96.7%	[4, 7], 85%	[9, 19], 96.7%
	forward	[2, 4], 92%	[9, 19], 96.7%	[2, 5, 7, 9], 87%	[9, 19], 96.7%
not normalized	backward	[1, 3, 4, 6, 9], 83%	[9, 19], 96.7%	[2], 85%	[9, 19], 96.7%
	forward	[2, 4], 92%	[9, 19], 96.7%	[2], 85%	[9, 19], 96.7%

KNN Classifier Ex3



There are improvements around k=3,4 after that the accuracy worsens.
The very small k =2 the accuracy also decreases. These effects are probably due to the nodes close to the boundary. The large K end up producing the same label for all instances making the search meaningless.
So small k are good for accuracy, bigger k make search meaningless.

K fold Validation Ex4

k	small	small	large	large personal
	forward	backward	forward	backward
1	[2, 5, 7, 9], 87%	[4, 7], 85%	[9, 19], 96.7%	[9, 19], 96.7%
2	[5, 7], 95%	[1], 82%	[9, 14, 19], 96%	[0, 1, 2, 3, 5, 7, 8, 10, 13, 16, 24, 25, 30, 32, 34, 37, 39], 81.80000000000001%
4	[4, 5, 7], 93%	[0, 1, 4, 8], 84%	[9, 19], 94.4%	[7, 19], 83.6%
5	[4, 5, 7], 89%	[1, 3, 4, 9], 83%	[9, 19], 94.5%	[0, 1, 2, 3, 4, 5, 8, 11, 13, 14, 16, 20, 26, 28, 30, 33, 34, 35, 37, 39], 80.8%
10	[2], 84%	[0, 1, 2, 3, 5], 82%	[9, 19], 94.1%	[5, 6, 7, 12, 13,

				14, 15, 16, 17, 18, 19, 20, 24, 25, 28, 32, 33, 34, 35, 39], 80.7%
20	[9], 76%	[1, 3, 4], 76%	[9, 19], 91.8%	[0, 1, 3, 4, 5, 7, 13, 15, 16, 17, 23, 32, 33, 34, 37, 39], 81.3%
25	[7], 71%	[1, 5, 6, 7, 8, 9], 73%	[9, 19], 89.1%	[4, 5, 8, 17, 20, 23, 28, 33, 34, 35, 37, 39], 79.2%
50	[9], 44%	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 44%	[9, 19], 89.5%	[3, 4, 7, 8, 11, 13, 17, 18, 24, 26, 27, 30, 33, 34, 35, 36, 37], 80.9%

From the data it's clear that these only work well on large datasets with forward search, with greater k improving speed. Backwards elimination was unable to reliably remove bad features. It might be improved by randomizing dataset order.

Conclusion

In the end both backward and forward perform similarly, with large data sets resulting in higher accuracy compared to small ones. Normalization didn't have much of an impact. K NN classification was improved only with k=3,4, with higher k being bad. K-fold verification only worked with forward search on large datasets.

Trace of your small dataset

```
Welcome to Timofey Malko(862311452) Feature Selection Algorithm.
Type the number of the algorithm you want to run.

    Forward Selection
    Backward Elimination.
1
Using no features and “random” evaluation, I get an accuracy of 0%

Beginning search.

Using feature(s) [0] accuracy is 73%
Using feature(s) [1] accuracy is 83%
Using feature(s) [2] accuracy is 85%
Using feature(s) [3] accuracy is 76%
Using feature(s) [4] accuracy is 75%
Using feature(s) [5] accuracy is 84%
Using feature(s) [6] accuracy is 75%
Using feature(s) [7] accuracy is 77%
Using feature(s) [8] accuracy is 71%
Using feature(s) [9] accuracy is 75%

Feature set [2] was best, accuracy is 85%

Using feature(s) [0, 2] accuracy is 78%
Using feature(s) [1, 2] accuracy is 81%
Using feature(s) [2, 3] accuracy is 78%
Using feature(s) [2, 4] accuracy is 75%
Using feature(s) [2, 5] accuracy is 78%
Using feature(s) [2, 6] accuracy is 79%
Using feature(s) [2, 7] accuracy is 81%
Using feature(s) [2, 8] accuracy is 81%
Using feature(s) [2, 9] accuracy is 81%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 2, 9] accuracy is 78%
Using feature(s) [1, 2, 9] accuracy is 80%
Using feature(s) [2, 3, 9] accuracy is 72%
Using feature(s) [2, 4, 9] accuracy is 82%
Using feature(s) [2, 5, 9] accuracy is 84%
Using feature(s) [2, 6, 9] accuracy is 75%
Using feature(s) [2, 7, 9] accuracy is 80%
Using feature(s) [2, 8, 9] accuracy is 82%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 2, 5, 9] accuracy is 76%
Using feature(s) [1, 2, 5, 9] accuracy is 79%
Using feature(s) [2, 3, 5, 9] accuracy is 75%
Using feature(s) [2, 4, 5, 9] accuracy is 79%
Using feature(s) [2, 5, 6, 9] accuracy is 80%
Using feature(s) [2, 5, 7, 9] accuracy is 87%
Using feature(s) [2, 5, 8, 9] accuracy is 80%

Feature set [2, 5, 7, 9] was best, accuracy is 87%
```

```
Using feature(s) [0, 2, 5, 7, 9] accuracy is 76%
Using feature(s) [1, 2, 5, 7, 9] accuracy is 83%
Using feature(s) [2, 3, 5, 7, 9] accuracy is 81%
Using feature(s) [2, 4, 5, 7, 9] accuracy is 82%
Using feature(s) [2, 5, 6, 7, 9] accuracy is 80%
Using feature(s) [2, 5, 7, 8, 9] accuracy is 86%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 2, 5, 7, 8, 9] accuracy is 76%
Using feature(s) [1, 2, 5, 7, 8, 9] accuracy is 74%
Using feature(s) [2, 3, 5, 7, 8, 9] accuracy is 76%
Using feature(s) [2, 4, 5, 7, 8, 9] accuracy is 80%
Using feature(s) [2, 5, 6, 7, 8, 9] accuracy is 80%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 2, 5, 6, 7, 8, 9] accuracy is 74%
Using feature(s) [1, 2, 5, 6, 7, 8, 9] accuracy is 72%
Using feature(s) [2, 3, 5, 6, 7, 8, 9] accuracy is 72%
Using feature(s) [2, 4, 5, 6, 7, 8, 9] accuracy is 77%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 2, 4, 5, 6, 7, 8, 9] accuracy is 73%
Using feature(s) [1, 2, 4, 5, 6, 7, 8, 9] accuracy is 74%
Using feature(s) [2, 3, 4, 5, 6, 7, 8, 9] accuracy is 72%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 1, 2, 4, 5, 6, 7, 8, 9] accuracy is 71%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 9] accuracy is 73%

(Warning, Accuracy has decreased!)%

Using feature(s) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] accuracy is 72%

(Warning, Accuracy has decreased!)%


Finished search!! The best feature subset is  [2, 5, 7, 9], which has an accuracy of 87%
```