Project 2: Feature Selection with Nearest Neighbor

Student Name1: Shaike Mukul SID: 862263999 Lecture Session: 001

Student Name2: Angelina Guzman SID: 862323157 Lecture Session: 001

Student Name3: Amina Penafiel SID: 862274322 Lecture Session: 001

Student Name4: Aryan Obrai SID: 862181380 Lecture Session: 001

Solution:

Not normalized:

Dataset	Best Feature Set	Accuracy
Small Number: 5	Forward Selection = {1, 2 ,4}	0.90
	Backward Elimination = {8, 10, 3, 4}	0.90
	Custom Algorithm = {}	NA
Large Number: 5	Forward Selection = {25, 21}	0.976
	Backward Elimination = {1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22, 26, 28, 29, 30, 31, 33, 34, 35, 37, 40}	0.797
	Custom Algorithm = {}	NA

Normalized:

Dataset	Best Feature Set	Accuracy
Small Number: 5	Forward Selection = {1, 2, 4}	0.90
	Backward Elimination = {8, 10, 3, 4}	0.90
	Custom Algorithm = {2, 4, 5, 6}	0.93

Large Number: 5	Forward Selection = {25, 21}	0.976
	Backward Elimination = {1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22, 26, 28, 29, 30, 31, 33, 34, 35, 37, 40}	0.797
	Custom Algorithm = {25, 26, 19, 21}	0.983

-------------------------------<Begin Report>----------------------------------

In completing this project, I consulted following resources:

https://sparrow.dev/numpy-norm/

https://docs.google.com/document/d/1oyfnJiUtEUca3cJLuMCs9WdW6MDL3lqW4Tte6p5bVWM/edit?usp=sharing

https://numpy.org/doc/stable/reference/generated/numpy.std.html

https://youtu.be/OUwfE_-tcfo

https://scikit-learn.org/stable/modules/cross_validation.html - for K-fold algorithm

Contribution of each student in the group:

Shaike: Forward selection code
Angelina: Backward elimination code, K-Fold
Amina: Lab Report
Aryan: Graphs and troubleshooting

I. Introduction

The goals of Project 2 are to learn about the nearest neighbor classifier, its sensitivity to irrelevant features, and how to make a feature search. In order to do this, we implemented greedy search, nearest neighbor classifier, leave-one-out validation, and feature search using all of these factors. This greedy search comes in the form of forward selection and backward elimination, where we used nearest neighbor and leave-one-out validation to find classification. From there we found the accuracy.

With this information, we searched for the relationship between forward selection and backward elimination to see how the nearest neighbor classifier reacts. Since the nearest neighbor is sensitive to irrelevant information, the way the features are added or removed to the feature set will change the accuracy. Throughout this project, we examine the results the different searches provide and how our code responds to our given datasets.

II. Challenges

One of our biggest challenges was trying to find a time when we could all meet and communicate what we want for the project. In the end, we all turned to Discord to message each other about our progress.

Another challenge was trying to understand each of the concepts given. A lot of these terms are new, and we're given very limited time to learn them. One of the best ways for us to learn was by using the internet to search for the concepts. One example would be learning to graph the different features.

An additional challenge we encountered was figuring out a custom algorithm. We ended up trying to implement K-fold validation using sklearn. But using the KFold() function also required looking for the right type of model to use, how many times to fold, etc. We tried implementing several models like Logistic Regression, Decision Tree, and then finally RandomForest. RandomForest seemed to give the best results which is why we went with that.

The other one was the implementation of the code and comparing it to the answers given. It's a bit of a struggle at first to get something similar, but once we understood that as long as it was similar and why, we had an idea that the code is okay.

The other challenge was translating code from the one given in the template into Python. While it worked out in the end, it took a while to translate.

III. Code Design

For our code design, we implement object-oriented programming, which makes it easier to collaborate.

<u>Object</u>

A feature selection object is constructed in __init__ with the following attributes:

values: data point values from the given dataset

labels: data point labels from the given dataset name: name of the user

n_features: number of features in the dataset

best_accuracy: the best accuracy given a subset of features in the feature set

Methods

read_file(): takes in the name of the dataset .txt file and reads its contents. It appends the labels and values in the file to the labels[] and values[] attributes to be used later.

leave_one_out_accuracy(self, labels, values, features, feature_to_add=None): in summary, this function predicts the accuracy of each data point based on the nearest-neighbor classification using the leave-one-out approach.

IV. 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

Plot some features and color code them by class and explore your dataset.

V. Algorithms

1. Forward Selection

The goal of forward selection is to get the highest percentage each time we add a new feature. With forward selection, we start off with an empty set. After that, we add one feature to this set and compare the sets with one feature by calculating its accuracy. This accuracy is found by calculating the correct instances over all instances for the dataset given. We take the set with the highest accuracy, then use that as the foundation to add a new feature, and then compare the accuracy of the set with the other sets. Find the highest accuracy, then keep repeating with the highest accuracy of the sets with the same number. Keep going until there are no new features and store the set with the highest accuracy overall.

2. **Backward Elimination**
Backward elimination starts with a set that has all the features in it. It checks the accuracy by calculating the correct instances over all the instances. Then it removes a feature from the set, then checks its accuracy. It will repeat this step until all features have been removed once, and then choose the set with the highest accuracy. Then it will repeat that part with the set with the highest accuracy as the basis. This continues until the whole set is empty. We also compare the set with the highest accuracy compared to the rest and see which one has the highest accuracy overall.

3. **K-Fold Cross Validation**
For K-Fold cross validation, the model is trained based on k-1 of the folds and uses the remaining data to check the accuracy of the model's prediction. The accuracy of k-fold cross validation is then just the average of the values computed for each fold being left out. The method below is how we implemented K-Fold Cross Validation.

k_fold_cross_validation(self, labels, values, features, feature_to_add=None): for this function, we used sklearn to import the KFold() method which takes in the number of splits to fold the data into, whether to shuffle the data or not, and a random state. We used 5 folds, because we thought it would work the best with the small and large datasets. We also decided to shuffle the data, just because it said that doing this would help decrease bias. We set a random state to 10, the 10 being an arbitrary value (changing this did nothing to our results), but it made sure that we produced the same accuracy every time. The KFold method returns iterators to where the data should be folded.

In combination with this, we also used cross_val_scores() which takes in a model, the current data values, the data labels as our target variable, and cv which is how many times we split the data. The model we used was the RandomForestClassifier, which uses decision trees to help predict the accuracy. Other models could have been used here like Decision Tree or Logistic Regression or Naive Bayes but RandomForestClassifier seemed like the simplest/easiest. Additionally, some models like Logistic Regression just didn't work for our data, we think because of a lack of variation. We set cv equal to our result from the KFold method. Then the function stores the scores of each of the folds into scores and we return the average value of these scores back to our special algorithm (which in this case, we just used forward selection but instead of leaving one out accuracy, we used k-fold).

VI. Analysis

Experiment 1: Comparing Forward Selection vs Backward Elimination.

<do this experiment for sure>

Compare accuracy with no feature selection vs with feature selection.

Compare feature set and accuracy for forward selection vs backward elimination. Talk about pros and cons of both algorithms.

Dataset	Best Feature Set	Accuracy
Small Number: 5	Forward Selection = {1, 2, 4}	0.90
	Backward Elimination = {8, 10, 3, 4}	0.90
	Custom Algorithm = {2, 4, 5, 6}	0.93
Large Number: 5	Forward Selection = {25, 21}	0.976
	Backward Elimination = {1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22, 26, 28, 29, 30, 31, 33, 34, 35, 37, 40}	0.797
	Custom Algorithm = {25, 26, 19, 21}	0.983

When doing this experiment, we realized how different backward elimination and forward selection are. Even though when the sets were smaller, they had a similar accuracy. They tended to have different accuries when the datasets were large as shown in the table with around 0.976 for forward selection and 0.797 for backward elimination. Compared to the default rate of around 0.30, they both fare better in the small datasets with a 0.9 accuracy. This may be because we are using "leave-one-out" for our accuracy, but it doesn't take into account the different features an instance may have.

With these features, some instances may have very similar features, even though they are different classes. With this in mind, backward elimination and forward selection tend to fare better than the default rate. Backward elimination and forward selection have their pros and cons. First and foremost, they are greedy algorithms, this helps to speed up the runtime, but they cannot consider all the cases are even the best case. They consider the best case for each of their instances.

Forward selection starts off with the empty set, adding one every time it finds the best accuracy for that part of the loop. It still goes through all of the features, but one-by-one. It may have a bit of a hard time comparing them. However, it tends to have higher rates and smaller feature sets than backward elimination.

Backward elimination has lower accuracy and bigger feature sets compared to forward selection. This may be because it starts off with all the features in the set, it has a hard time comparing to when it has less features because it doesn't check the features in the same order as forward selection. With this approach, it should have a better chance to not overfit all the data.

In all, forward selection, backward elimination, and "leave-one-out" all have its own purposes. The best way to use them is to figure out what is the best scenario for them.

Experiment 2: Effect of normalization

Compare accuracy when using normalized vs unnormalized data.

Not normalized:

Dataset	Best Feature Set	Accuracy
Small Number: 5	Forward Selection = {1, 2 ,4}	0.90
	Backward Elimination = {8, 10, 3, 4}	0.90
	Custom Algorithm = Not implemented	NA
Large Number: 5	Forward Selection = {25, 21}	0.976
	Backward Elimination = {1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22, 26, 28, 29, 30, 31, 33, 34, 35, 37, 40}	0.797
	Custom Algorithm = Not implemented	NA

Normalized:

Dataset	Best Feature Set	Accuracy
Small Number: 5	Forward Selection = {1, 2, 4}	0.90
	Backward Elimination = {8, 10, 3, 4}	0.90
	Custom Algorithm = Not implemented	NA

Large Number: 5	Forward Selection = {25, 21}	0.976
	Backward Elimination = {1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 15, 16, 17, 19, 20, 22, 26, 28, 29, 30, 31, 33, 34, 35, 37, 40}	0.797
	Custom Algorithm = Not implemented	NA

Normalizing data organizes the data to make them similar across all fields, which makes it easier to find the information to group and analyze. What we have done here, we normalized the data by finding its euclidean distance divided by the standard deviation of all the instances. However, when we did that, it doesn't seem there were any big differences between the normalized data and not normalized data. This may be the case that our percentages round up, but it shouldn't be the case.

In all, normalization can help with grouping different instances and improve accuracy. For this information, it did not change our results over both tables.

VII. Conclusion

General summary of your findings from the Analysis. Potential improvements to this approach of doing feature selection.

From our analysis of backward elimination and forward selection, they both do fine with smaller data sets, but backward elimination tends to have a lower accuracy compared to the latter. This may be because backward elimination starts off with all the features in the set and removes it one-by-one, by doing so, it misses out on features it deleted previously. However, both of these greedy algorithms have the same goal and different pros and cons depending on what the search needs.

On the other hand, normalization didn't do much for this set, even though it should help organize and sort the data. This may be because how our data is and what our algorithms for the searches were set up.

Potential improvements can be seen using our custom algorithms, using k-fold cross. This is another way to sort our instances and accuracies. Compared to the other two algorithms, this has a faster runtime than O(n^2), allowing us to quickly find a good feature set. Another way for improvement is comparing the searches, it may be farfetched, but comparing the accuracies between forward selection and backward elimination can reveal information about these instances. From there, we can tweak each algorithm to what the search needs.

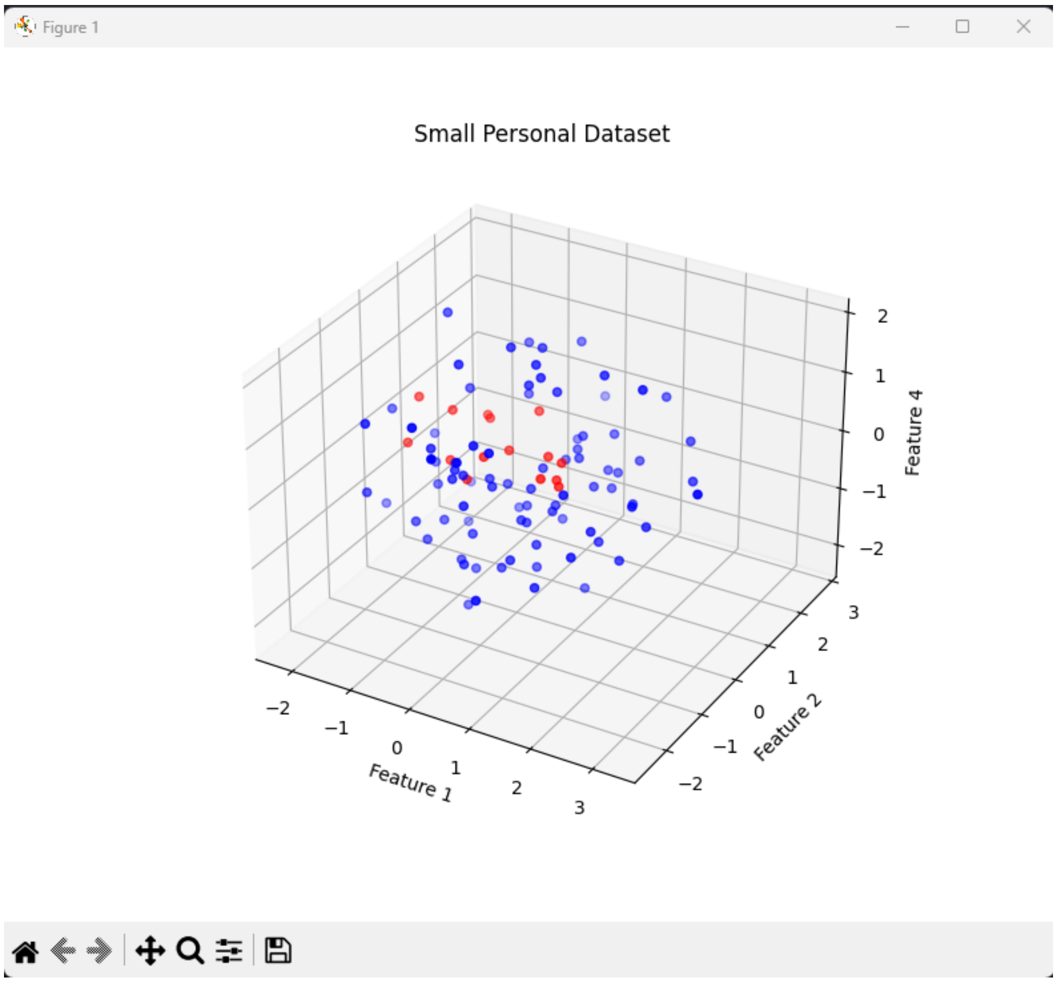
VIII. Trace of your small dataset

Foward Selection:

```
Welcome to Group 5's Feature Selection Algorithm.

Type the number of the algorithm you want to run.

1. Forward Selection

2. Backward Elimination

3. Group 5  Special Algorithm.

1

This dataset has 10 features (not including the class attribute), with 100 instances.


Running nearest neighbor with no features (default rate), using “leaving-one-out”
evaluation, I get an accuracy of 30.652%


Using feature(s) {1} accuracy is 74.0%

Using feature(s) {2} accuracy is 85.0%

Using feature(s) {3} accuracy is 66.0%

Using feature(s) {4} accuracy is 82.0%

Using feature(s) {5} accuracy is 72.0%

Using feature(s) {6} accuracy is 70.0%

Using feature(s) {7} accuracy is 71.0%

Using feature(s) {8} accuracy is 74.0%

Using feature(s) {9} accuracy is 73.0%

Using feature(s) {10} accuracy is 76.0%

Feature set {2} was best, accuracy is 85.0%



Using feature(s) {1, 2} accuracy is 81.0%

Using feature(s) {2, 3} accuracy is 74.0%

Using feature(s) {2, 4} accuracy is 89.0%
Using feature(s) {2, 5} accuracy is 73.0%
Using feature(s) {2, 6} accuracy is 77.0%
Using feature(s) {2, 7} accuracy is 75.0%
Using feature(s) {8, 2} accuracy is 77.0%
Using feature(s) {9, 2} accuracy is 81.0%
Using feature(s) {2, 10} accuracy is 81.0%
Feature set {2, 4} was best, accuracy is 89.0%


Using feature(s) {1, 2, 4} accuracy is 90.0%
Using feature(s) {2, 3, 4} accuracy is 85.0%
Using feature(s) {2, 4, 5} accuracy is 84.0%
Using feature(s) {2, 4, 6} accuracy is 88.0%
Using feature(s) {2, 4, 7} accuracy is 82.0%
Using feature(s) {8, 2, 4} accuracy is 86.0%
Using feature(s) {9, 2, 4} accuracy is 86.0%
Using feature(s) {2, 10, 4} accuracy is 85.0%
Feature set {1, 2, 4} was best, accuracy is 90.0%


Using feature(s) {1, 2, 3, 4} accuracy is 83.0%
Using feature(s) {1, 2, 4, 5} accuracy is 86.0%
Using feature(s) {1, 2, 4, 6} accuracy is 85.0%
Using feature(s) {1, 2, 4, 7} accuracy is 85.0%
Using feature(s) {8, 1, 2, 4} accuracy is 88.0%
Using feature(s) {1, 2, 4, 9} accuracy is 86.0%
Using feature(s) {1, 2, 10, 4} accuracy is 83.0%
```

```
Feature set {8, 1, 2, 4} was best, accuracy is 88.0%



Using feature(s) {1, 2, 3, 4, 8} accuracy is 85.0%
Using feature(s) {1, 2, 4, 5, 8} accuracy is 89.0%
Using feature(s) {1, 2, 4, 6, 8} accuracy is 85.0%
Using feature(s) {1, 2, 4, 7, 8} accuracy is 86.0%
Using feature(s) {1, 2, 4, 8, 9} accuracy is 90.0%
Using feature(s) {1, 2, 4, 8, 10} accuracy is 85.0%
Feature set {1, 2, 4, 8, 9} was best, accuracy is 90.0%



Using feature(s) {1, 2, 3, 4, 8, 9} accuracy is 81.0%
Using feature(s) {1, 2, 4, 5, 8, 9} accuracy is 77.0%
Using feature(s) {1, 2, 4, 6, 8, 9} accuracy is 80.0%
Using feature(s) {1, 2, 4, 7, 8, 9} accuracy is 75.0%
Using feature(s) {1, 2, 4, 8, 9, 10} accuracy is 85.0%
Feature set {1, 2, 4, 8, 9, 10} was best, accuracy is 85.0%



Using feature(s) {1, 2, 3, 4, 8, 9, 10} accuracy is 86.0%
Using feature(s) {1, 2, 4, 5, 8, 9, 10} accuracy is 80.0%
Using feature(s) {1, 2, 4, 6, 8, 9, 10} accuracy is 80.0%
Using feature(s) {1, 2, 4, 7, 8, 9, 10} accuracy is 78.0%
Feature set {1, 2, 3, 4, 8, 9, 10} was best, accuracy is 86.0%
```

```
Using feature(s) {1, 2, 3, 4, 5, 8, 9, 10} accuracy is 82.0%

Using feature(s) {1, 2, 3, 4, 6, 8, 9, 10} accuracy is 83.0%

Using feature(s) {1, 2, 3, 4, 7, 8, 9, 10} accuracy is 83.0%

Feature set {1, 2, 3, 4, 6, 8, 9, 10} was best, accuracy is 83.0%



Using feature(s) {1, 2, 3, 4, 5, 6, 8, 9, 10} accuracy is 77.0%

Using feature(s) {1, 2, 3, 4, 6, 7, 8, 9, 10} accuracy is 76.0%

Feature set {1, 2, 3, 4, 5, 6, 8, 9, 10} was best, accuracy is 77.0%



Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} accuracy is 75.0%

Feature set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} was best, accuracy is 75.0%



Finished search!! The best feature subset is {1, 2, 4}, which has an accuracy of 90.0%
```



*the different colors are here just to show depth (all blues are one class, and all reds are the other class)

Backward Elimination:

This dataset has 10 features (not including the class attribute), with 100 instances.

Running nearest neighbor with no features (default rate), using "leaving-one-out" evaluation, I get an accuracy of 6.938%

Using feature(s) {2, 3, 4, 5, 6, 7, 8, 9, 10}, accuracy is 76.0%

Using feature(s) {1, 3, 4, 5, 6, 7, 8, 9, 10}, accuracy is 76.0%

```
Using feature(s) {1, 2, 4, 5, 6, 7, 8, 9, 10}, accuracy is 71.0%

Using feature(s) {1, 2, 3, 5, 6, 7, 8, 9, 10}, accuracy is 72.0%

Using feature(s) {1, 2, 3, 4, 6, 7, 8, 9, 10}, accuracy is 76.0%

Using feature(s) {1, 2, 3, 4, 5, 7, 8, 9, 10}, accuracy is 76.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 8, 9, 10}, accuracy is 77.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 7, 9, 10}, accuracy is 74.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 10}, accuracy is 82.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8, 9}, accuracy is 71.0%

Feature set {1, 2, 3, 4, 5, 6, 7, 8, 10} was best, accuracy is 82.0%


Using feature(s) {2, 3, 4, 5, 6, 7, 8, 10}, accuracy is 83.0%

Using feature(s) {1, 3, 4, 5, 6, 7, 8, 10}, accuracy is 80.0%

Using feature(s) {1, 2, 4, 5, 6, 7, 8, 10}, accuracy is 78.0%

Using feature(s) {1, 2, 3, 5, 6, 7, 8, 10}, accuracy is 77.0%
Using feature(s) {1, 2, 3, 4, 6, 7, 8, 10}, accuracy is 77.0%

Using feature(s) {1, 2, 3, 4, 5, 7, 8, 10}, accuracy is 89.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 8, 10}, accuracy is 84.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 7, 10}, accuracy is 81.0%

Using feature(s) {1, 2, 3, 4, 5, 6, 7, 8}, accuracy is 76.0%

Feature set {1, 2, 3, 4, 5, 7, 8, 10} was best, accuracy is 89.0%


Using feature(s) {2, 3, 4, 5, 7, 8, 10}, accuracy is 87.0%

Using feature(s) {1, 3, 4, 5, 7, 8, 10}, accuracy is 82.0%

Using feature(s) {1, 2, 4, 5, 7, 8, 10}, accuracy is 83.0%

Using feature(s) {1, 2, 3, 5, 7, 8, 10}, accuracy is 78.0%

Using feature(s) {1, 2, 3, 4, 7, 8, 10}, accuracy is 83.0%

Using feature(s) {1, 2, 3, 4, 5, 8, 10}, accuracy is 86.0%

Using feature(s) {1, 2, 3, 4, 5, 7, 10}, accuracy is 83.0%
Using feature(s) {1, 2, 3, 4, 5, 7, 8}, accuracy is 77.0%

Feature set {2, 3, 4, 5, 7, 8, 10} was best, accuracy is 87.0%


Using feature(s) {3, 4, 5, 7, 8, 10}, accuracy is 86.0%

Using feature(s) {2, 4, 5, 7, 8, 10}, accuracy is 83.0%

Using feature(s) {2, 3, 5, 7, 8, 10}, accuracy is 75.0%

Using feature(s) {2, 3, 4, 7, 8, 10}, accuracy is 82.0%

Using feature(s) {2, 3, 4, 5, 8, 10}, accuracy is 85.0%

Using feature(s) {2, 3, 4, 5, 7, 10}, accuracy is 83.0%

Using feature(s) {2, 3, 4, 5, 7, 8}, accuracy is 81.0%

Feature set {3, 4, 5, 7, 8, 10} was best, accuracy is 86.0%


Using feature(s) {4, 5, 7, 8, 10}, accuracy is 83.0%

Using feature(s) {3, 5, 7, 8, 10}, accuracy is 82.0%

Using feature(s) {3, 4, 7, 8, 10}, accuracy is 83.0%
Using feature(s) {3, 4, 5, 8, 10}, accuracy is 86.0%

Using feature(s) {3, 4, 5, 7, 10}, accuracy is 82.0%

Using feature(s) {3, 4, 5, 7, 8}, accuracy is 82.0%

Feature set {3, 4, 5, 8, 10} was best, accuracy is 86.0%


Using feature(s) {8, 10, 4, 5}, accuracy is 81.0%
Using feature(s) {8, 10, 3, 5}, accuracy is 76.0%
Using feature(s) {8, 10, 3, 4}, accuracy is 90.0%
Using feature(s) {10, 3, 4, 5}, accuracy is 81.0%
Using feature(s) {8, 3, 4, 5}, accuracy is 81.0%
Feature set {8, 10, 3, 4} was best, accuracy is 90.0%

Using feature(s) {8, 10, 4}, accuracy is 76.0%
Using feature(s) {8, 10, 3}, accuracy is 78.0%
Using feature(s) {10, 3, 4}, accuracy is 81.0%
Using feature(s) {8, 3, 4}, accuracy is 83.0%
Feature set {8, 3, 4} was best, accuracy is 83.0%


Using feature(s) {8, 4}, accuracy is 84.0%
Using feature(s) {8, 3}, accuracy is 75.0%
Using feature(s) {3, 4}, accuracy is 78.0%
```

Feature set {8, 4} was best, accuracy is 84.0%

Using feature(s) {8}, accuracy is 74.0%

Using feature(s) {4}, accuracy is 82.0%

Feature set {4} was best, accuracy is 82.0%

Finished search!! The best feature subset is {8, 10, 3, 4}, which has an accuracy of 90.0%

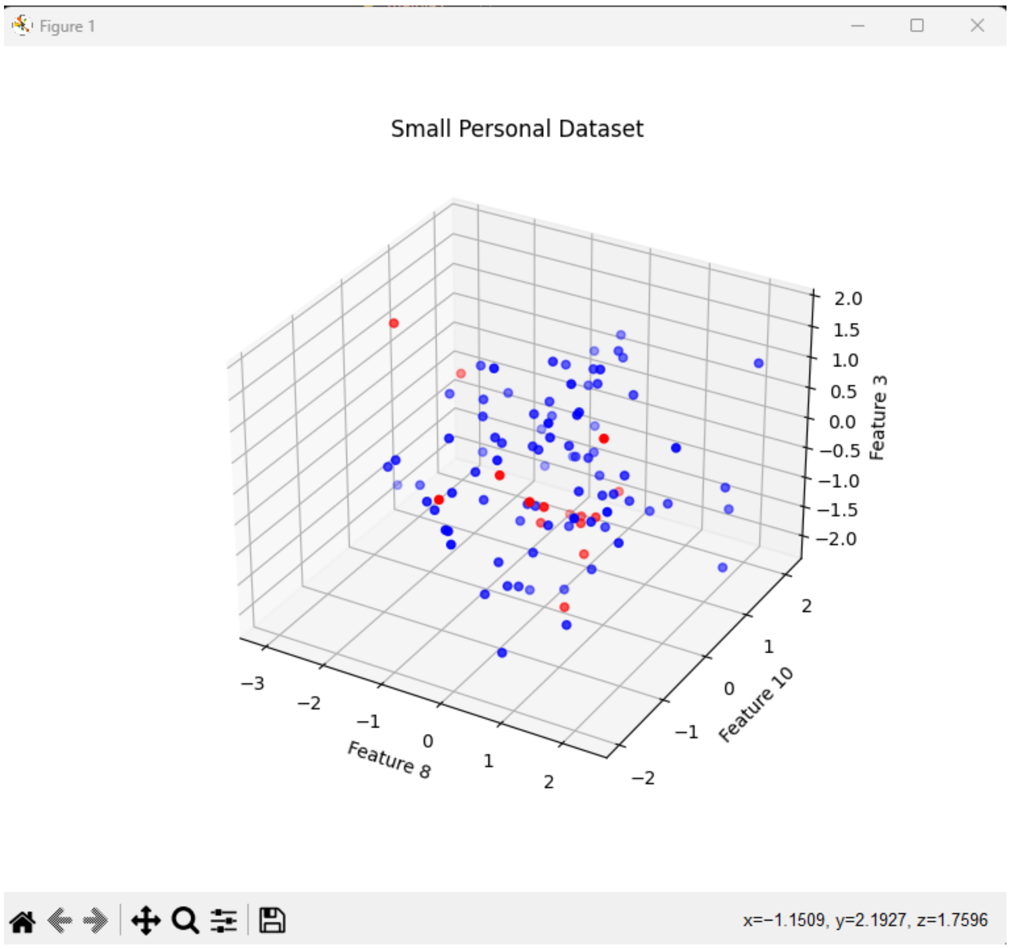


Figure 1
Small Personal Dataset
Feature 8
Feature 10
Feature 3
x=−1.1509, y=2.1927, z=1.7596