Jacques Fracchia
862010872
UCR CS170

Project #2: Feature Selection with Nearest Neighbor

Challenges:

To begin with, the data was very difficult to parse and set up in arrays. Initially I was using C++, but found that to be very tedious. After looking on stackoverflow, seeing so many python solutions is what made me decide to do this project in python. When calling functions in forward search and backwards search I found that data from the array would be lost when passed into the other function. This lead to a lot of trouble and also to the conclusion that I need to create a deep copy of the array and pass in a temporary array instead. Overall, the challenges were getting used to coding in python and how the file structure will look like to completely separate the leave one out validator and the search algorithms. The professors notes on the project helped tremendously!

Structure:

main.py was six functions including main. Main reads in a passed in file of our dataset and then immediately normalizes it. This allows us to see the differences in numbers easier and allows us to classify the data. It then gives an overall accuracy using nearest neighbor, then prompts the user to enter if they want to use forward selection or backwards selection search algorithms.

Forward and Backward selection then iterates through every possible combination, either starting with the full set in an array in backwards selection, or an empty set in forward selection. The algorithm iterates through each possible combination of features before adding or subtracting in another feature. At each step or level the algorithm calculates the accuracy for each option using the leaving-one-out validator algorithm. After each iteration, the algorithm selects the highest accuracy from each possible option then restarts its next iteration with that new and updated current array. The function keeps track of the over all highest accuracy combination of features and returns the final highest accuracy and the combination before termination.

Analysis:

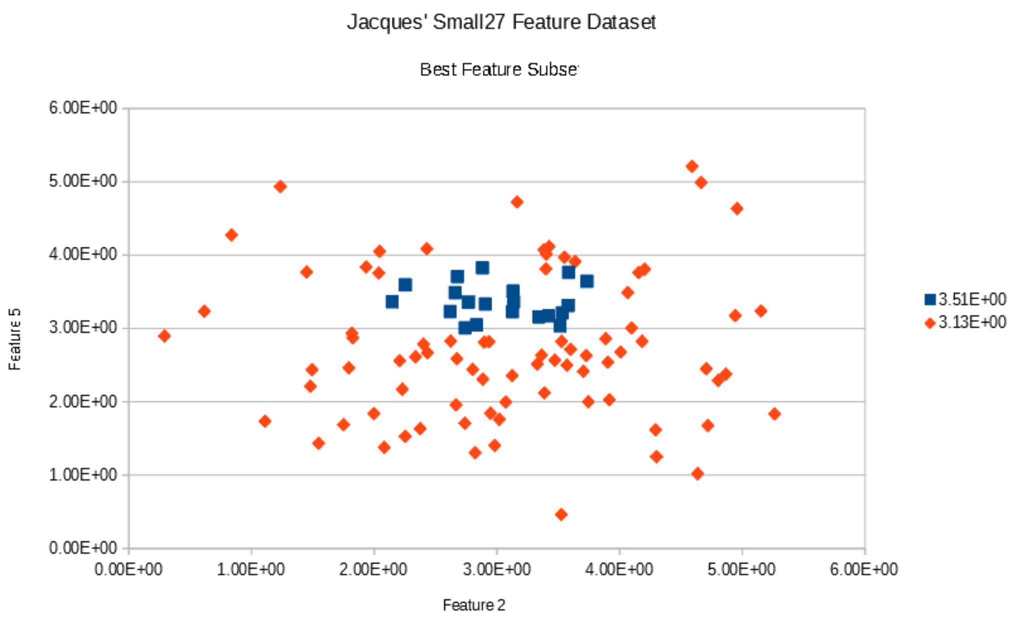

Figure 1: Best Feature Subset with Forward Selection for cs_170_small27.txt

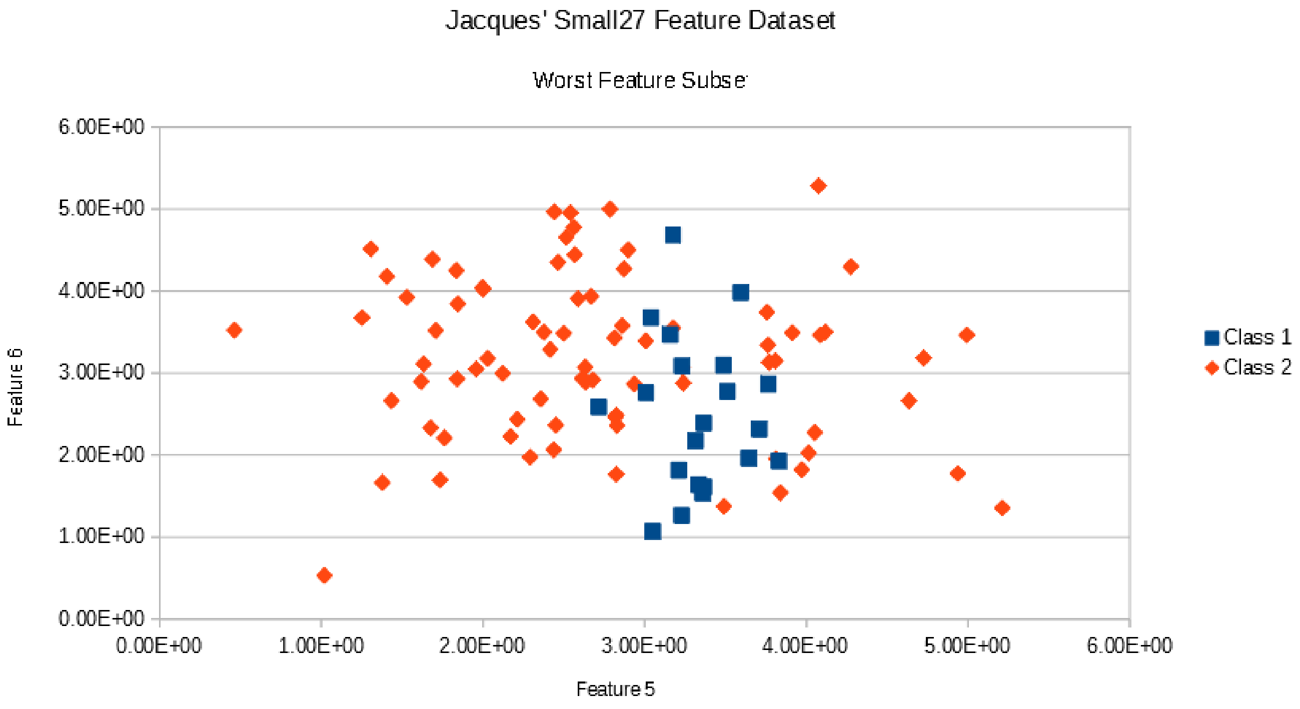

Figure 2: Worst Feature Subset with Forward Selection for cs_170_small27.txt

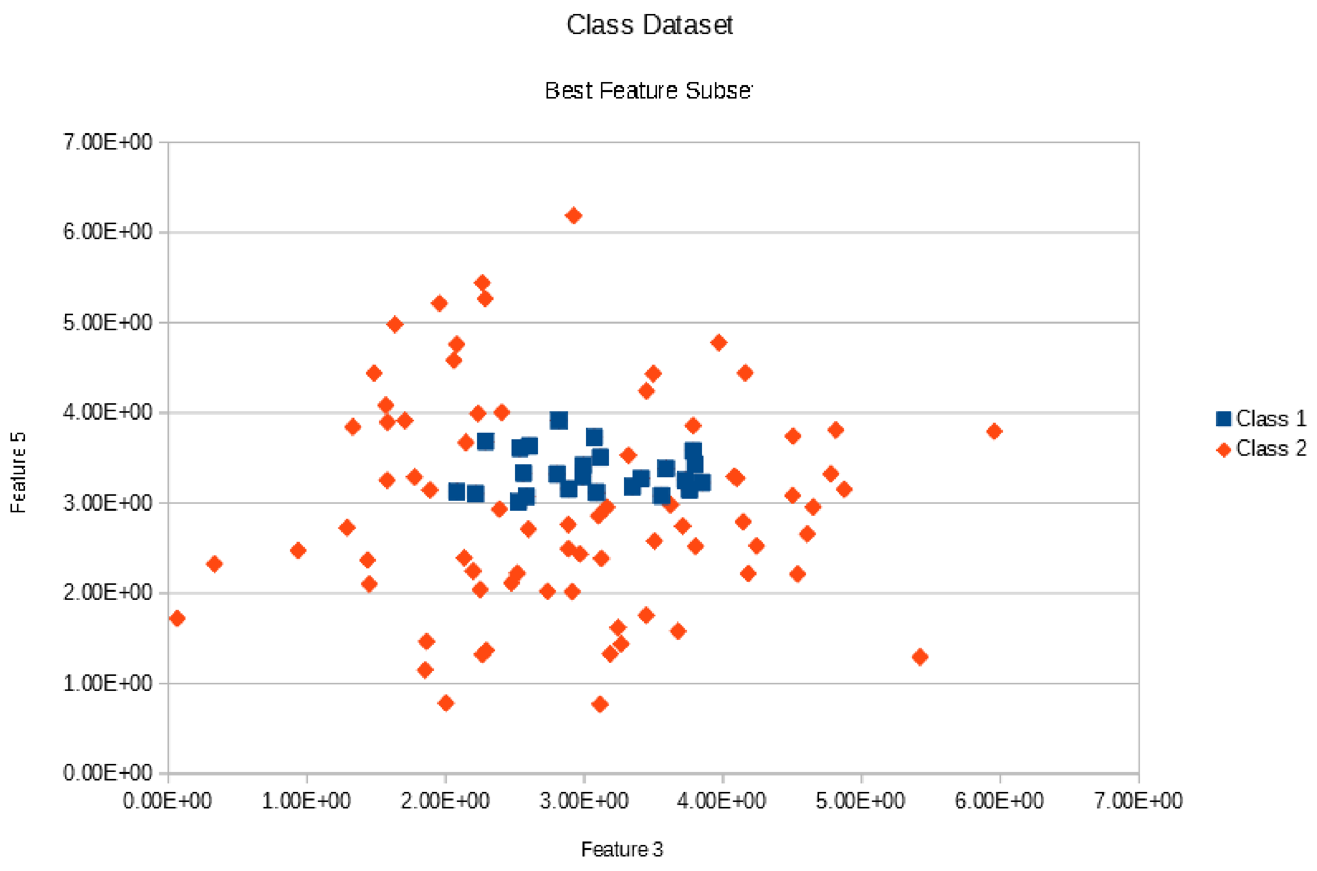

Figure 3: Best Feature Subset with Forward Selection for cs_170_small80.txt

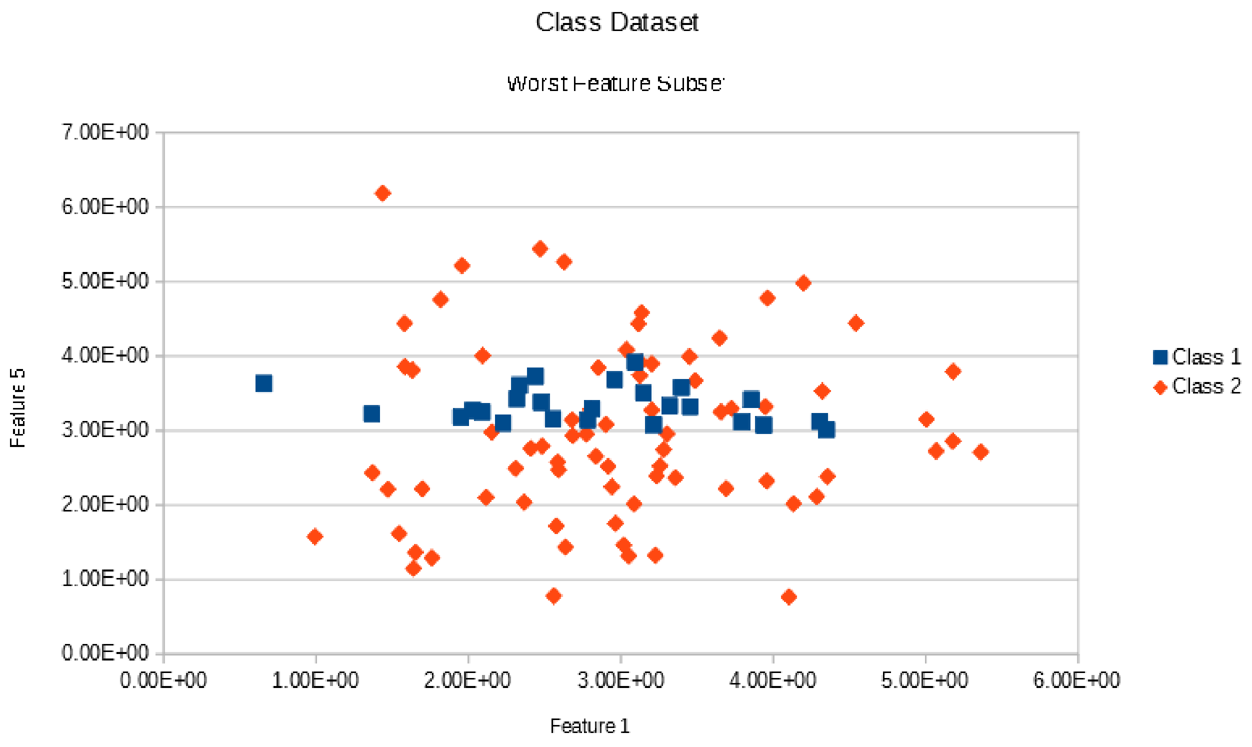

Figure 4: Worst Feature Subset with Forward Selection for cs_170_small80.txt

Results:

File	Forward Selection – Best Subset	Backwards Selection – Best Subset
cs_170_small27.txt	[2, 5] – 95.95%	[5, 10] – 84.84%
cs_170_large27.txt	[31, 35] – 97.597%	[1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 16, 18, 20, 26, 27, 28, 29, 30, 31, 34, 35, 37, 38] – 86.386%

We can clearly see from figure 1 and figure 2 why one combination [2, 5] is so much better than the worst combination [5, 6]. In figure 1 hardly any of the features from the two classes are overlapping. This allows us to differentiate between the two classes very easily when comparing these two features. On the other hand, looking at figure 2, all of the points are overlapping each other. It would be impossible, if given features 4 and 5 to differentiate which class is which. This is also true for the dataset small80 shown in figures 3 and 4. We can clearly see which classes are separated and not overlapping in figure 3, where in figure 4 everything is overlapping.

This is why our algorithms select the highest accuracy at every step, then expand on that until we have exhausted all combinations of features. Even though the two algorithms work very similar, it is evident from the result table that forward selection works slightly better than backwards selection. This statement would need to be tested further with more data to see if true but for my personalized data set Forward selection outperformed backwards selection.

When trying to find the best combination of features to use to identify different types of classes, using forward selection and backwards selection are both very powerful and will provide a high accuracy with an efficient combination of features to classify data sets.

Code:

```python
import math

import copy

"""
Written using python version 2.7
Jacques Fracchia
CS170 UCR
"""

def ForwardSelection(data, num_instances, num_features):
        final_set_of_features = []
        current_set_of_features = []
        final_accuracy = 0.0

        for i in range(num_features):
                feature_to_add_at_this_level = -1
                feature_to_add = -1
                best_so_far_accuracy = 0.0

                for j in range(1, num_features+1):
                        if j not in current_set_of_features:
                                temp_features = copy.deepcopy(current_set_of_features)
                                temp_features.append(j)

                                accuracy = leave_one_out_cross_validation(data, temp_features, num_instances)
                                print "\tUsing feature(s) ", temp_features, " accuracy is ", accuracy, "%"

                                if accuracy > final_accuracy:
                                        final_accuracy = accuracy
                                        feature_to_add_at_this_level = j

                                if accuracy > best_so_far_accuracy:
                                        best_so_far_accuracy = accuracy
                                        feature_to_add = j

                if(feature_to_add_at_this_level >= 0):
                        current_set_of_features.append(feature_to_add_at_this_level)
                        final_set_of_features.append(feature_to_add_at_this_level)
                        print "\n\nFeature set ", current_set_of_features, " was best, accuracy is ", final_accuracy,
"%\n\n"

                else:
                        print "\n\n(Warning, Accuracy has decreased! Continuing search in case of local maxima)"

                        current_set_of_features.append(feature_to_add)
                        print "Feature set ", current_set_of_features, " was best, accuracy is ",
best_so_far_accuracy, "%\n\n"

        print "Finished search!! The best feature subset is", final_set_of_features, " which has an accuracy of
accuracy: ", final_accuracy, "%"


def BackwardElimination(data, num_instances, num_features):
        current_set_of_features = []
        final_set_of_features = []
        final_accuracy = 0.0

        for i in range(1, num_features+1):
                current_set_of_features.append(i)
                final_set_of_features.append(i)

        for i in range(num_features):
                feature_to_remove_at_this_level = -1
                feature_to_remove = -1
                best_so_far_accuracy = 0.0

                for j in range(1, num_features + 1):
                        if j in current_set_of_features:
                                temp_subset = copy.deepcopy(current_set_of_features)
                                temp_subset.remove(j)

                                accuracy = leave_one_out_cross_validation(data, temp_subset, num_instances)
                                print "\tUsing feature(s) ", temp_subset, " accuracy is ", accuracy, "%"

                                if accuracy > final_accuracy:
                                        final_accuracy = accuracy
                                        feature_to_remove_at_this_level = j

                                if accuracy > best_so_far_accuracy:
                                        best_so_far_accuracy = accuracy
                                        feature_to_remove = j

                if feature_to_remove_at_this_level >= 0:
                        current_set_of_features.remove(feature_to_remove_at_this_level)
                        final_set_of_features = current_set_of_features[:]
                        print "\n\nFeature set ", current_set_of_features, " was best, accuracy is ", final_accuracy,
"%\n\n"

                else:
                        print "\n\n(Warning, Accuracy has decreased! Continuing search in case of local maxima)"

                        current_set_of_features.remove(feature_to_remove)
                        print "Feature set ", current_set_of_features, " was best, accuracy is ",
best_so_far_accuracy, "%\n\n"


        print "Finished search!! The best feature subset is", final_set_of_features, " which has an accuracy of
accuracy: ", final_accuracy, "%"
```

```
def Normalize(dataset, features_len, instances_len):
	std = []
	mean = []

	for i in range(features_len):
		mean.append((sum(row[i] for row in dataset))/instances_len)
		variance = sum( pow((row[i] - mean[i-1]), 2) for row in dataset)/instances_len
		std.append(math.sqrt(variance))

	for i in range(instances_len):
		for j in range (1, features_len + 1):
			x = dataset[i][j] - mean[j-1]
			dataset[i][j] = x / std[j-1]

	return dataset

def NearestNeighbor(dataset, position, features, instances_len):
	result = 0
	closest_path = float("inf")

	for i in range(instances_len):
		if position == i:
			pass

		else:
			length = 0
			for j in range(len(features)):
				x = dataset[i][features[j]] - dataset[position][features[j]]
				length = length + pow(x, 2)

			length = math.sqrt(length)
			if length < closest_path:
				closest_path = length
				result = i

	return result

def leave_one_out_cross_validation(dataset, features, instances_len):
	valid = 0.0
	for i in range(instances_len):
		position = i
		nearest_neighbor = NearestNeighbor(dataset, position, features, instances_len)

		if dataset[nearest_neighbor][0] == dataset[position][0]:
			valid = valid + 1

	accuracy = (valid / instances_len) * 100
	return accuracy


def main():
	print "Welcome to Jacques Fracchia\'s Feature Selection Algorithm."
	text = raw_input("Type in the name of the file to test: ")
	try:
		dataset = open(text, "r")
	except:
		raise IOError("Could not read: " + dataset)

	get_line = dataset.readline()
	instances_len = sum(1 for line in dataset)

	dataset.seek(0)

	instances = [[] for i in range (instances_len)]
	for i in range(instances_len):
		instances[i] = [float(x) for x in dataset.readline().split()]

	dataset.seek(0)

	features_len = len(get_line.split()) - 1
	features = []
	for i in range(0, features_len):
		features.append(i)

	dataset.seek(0)
	print "This dataset has " + str(features_len) + " features (not including the class attribute), with " +
str(instances_len) + " instances."
	print "Please wait while I normalize the data... Done!"
	normalize_instances = Normalize(instances, features_len, instances_len)

	print "Type the number of the algorithm you want to run."
	print "1) Forward Selection"
	print "2) Backward Elimination"
	selection = int(raw_input())
	accuracy = leave_one_out_cross_validation(normalize_instances, features, instances_len)

	print "Running nearest neighbor with all " + str(features_len) + " features, using \"leaving-one-out\"
evaluation, I get an accuracy of " + str(accuracy) + "%."
	print "\nBeginning search.\n"
	if selection == 1:
		ForwardSelection(normalize_instances, instances_len, features_len)
	elif selection == 2:
		BackwardElimination(normalize_instances, instances_len, features_len)

if __name__ == "__main__":
	main()
```