# CS 170 - Introduction to Artificial Intelligence

Project 2: Feature Selection

Kevin Kim
861168574
3/18/17

**Introduction**

In this project we explored the subject of feature selection by using the nearest neighbor algorithm. We received datasets which consisted of instances of either class 1 or class 2. Each class had a certain amount of features that classified it as that class. The goal of this project was to use these datasets to find out which features were the best for classification. To solve this problem we used the nearest neighbor algorithm in conjunction with different tuples of features. We used two greedy methods of forward selection and backward elimination in order to select which features were best and construct a trace of the algorithm.

**The Program:**

**Classes**

I completed this programming assignment using c++. I started by creating an Instance class which had two data members: a class type and a vector of features. This class was responsible for handling each instance (row) in the dataset where the first column of the row was stored as the class type and all other columns were stored as features. Essentially what happened was, I would read in the data from the dataset using the fstream class and then parse the data by storing each row in an Instance object. I also created a Problem class which I visualized as the class that handled solving the problem. This class contained a vector of Instances, which were stored after the dataset of size n had been converted into n Instance objects.

**Data Structures**

I decided to use vectors for this assignment because I wanted to make my code as adaptable as possible. I didn't want to hardcode sizes of an array based on the size of the dataset. Instead I could use the vector push_back() function which meant I didn't have to initially allocate space that I might not use. I also created a Struct which I called Feature_Set. Feature_Set had two data members: an accuracy, and a vector(tuple) of features. Essentially, this struct allowed me to store features with their respective accuracy. This made tracing and also displaying the data much easier.

**The Algorithm:**

**Euclidian Distance**

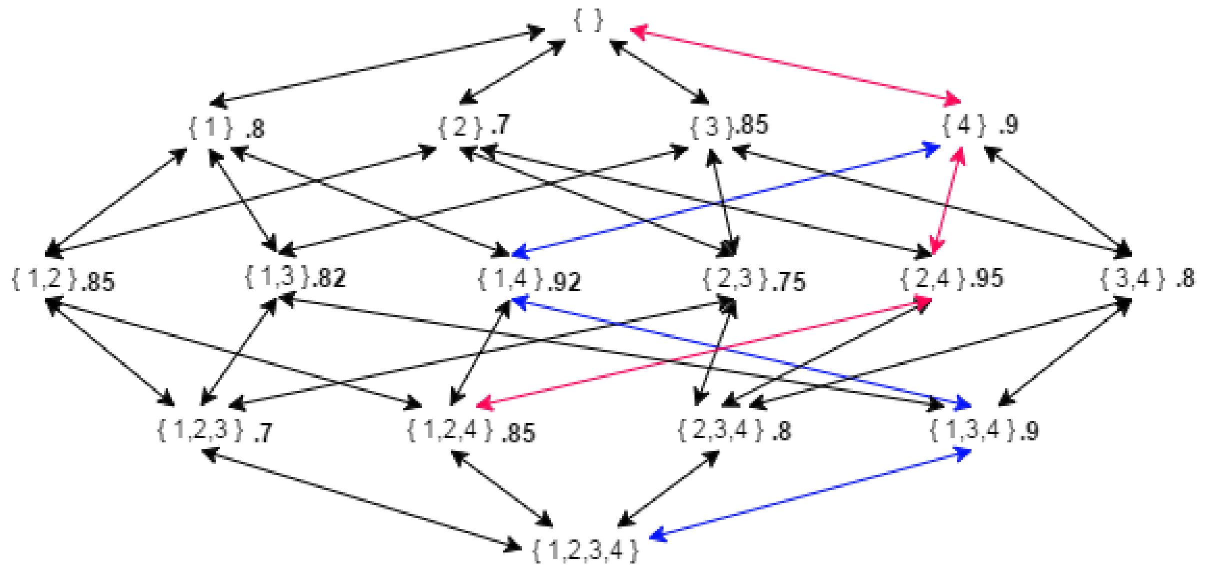
Since we were using the nearest neighbor algorithm, we had to implement the Euclidian distance formula. I included this as a helper function in my Problem class. I had the function take in two instances and a tuple of which features to use, and it returned the distance of the two instances based on the selected features.

**Nearest Neighbors**

I implemented the nearest neighbors algorithm in my Problem class. This function took in a vector of features and returned an accuracy based on those features. The Nearest neighbor algorithm used a nested for loop to cycle through the dataset and separate the dataset into test data and training data. For each test data iteration, the function used the Euclidian distance function to calculate the distance between each instance of training data while keeping track of the nearest neighbor. At the end, the nearest neighbor was compared to the test data and if there was a match in terms of class, then the count for correctness would increase. At the end of the algorithm, the function would return an accuracy for the set of features that were passed in.

**Forward Selection / Backward Elimination**

My forward selection and backward elimination functions were basically the same. The only difference was, for forward selection I started with an empty set of features and continued to add features until all the features were added. For backward elimination I did the opposite where I started with a full set of features and eliminated a feature until I reached the empty set. For both algorithms, we want to make the greedy choice. This means that at every step, whether we are adding a feature, or eliminating a feature, we want to make a choice that returns the highest accuracy. To do this, I used a loop that will only add or subtract features from the greedy choice, depending on the choice of algorithm. The problem with these algorithms being greedy is that they can get caught in local minima and maxima.



In this diagram I am showing how this might affect the overall results. The red arrows signify forward selection and the blue arrows signify backward elimination. As we can see, in forward selection, we add the features that give us the highest accuracy (I used arbitrary accuracies). This gives us a path of { } →{4} →{2,4}→{1,2,4} where the highest accuracy is .95 with feature set {2,4}. In backward elimination we subtract features to give us a set with the highest accuracy. This gives us {1,3,4}→{1,4}→{4} where the highest accuracy is .92 with a feature set {1,4}. As we can see the backward elimination doesn't yield the actual highest accuracy which is

.95 with feature set {2,4}. This is because backward elimination eliminates feature 2 early on and thus has no way to get out of its local search.

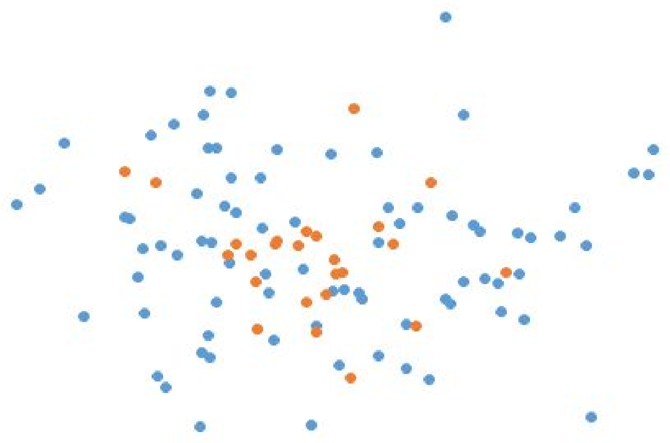
**Data and Analysis:**

| Dataset | Algorithm | Best accuracy found | Best features found | runtime |
|---|---|---|---|---|
| Small dataset | Forward | 0.92 | { 5,3 } | 1 sec |
| | Backward | 0.83 | { 2,4,5,7,10 } | 1 sec |
| Large dataset | Forward | 0.955 | { 27,1 } | 3 min 20 sec |
| | Backward | 0.847 | { 27 } | 4 min 10 sec |
| Custom small dataset | Forward | 0.95 | { 9,1 } | 1 sec |
| | Backward | 0.95 | { 9,1 } | 1 sec |
| Custom large dataset | Forward | 0.97 | { 20,40 } | 3 min 17 sec |
| | Backward | 0.856 | { 12,20 } | 4 min |

Looking at the table, we can see how the forward and backward algorithms differ in their ability to find the best feature set and accuracy. In my small custom test, both forward and backward search returned the same results, but this was the only case. My program was not as fast as it could have been, however runtime was not one of my chief concerns. Interestingly, most of the results, with the exception of one, had two features in the best feature set and one feature set only had one feature. Obviously one feature is not very reliable because the nearest neighbor algorithm is susceptible to outlier values. Thus it makes sense to have more than one feature in the feature set.

I made my program run until all we had either a feature set of all features (forward) or a feature set of no features (backward). In most cases the value that was returned was the value of the local maximum. However, in many of the backward runs, the maximum value the program found was after a decrease in accuracy. The value dipped and then rose as less features were considered. This is also why I think forward selection was better than backward elimination. Backward elimination for the most part didn't find the maximum accuracy and also was easily trapped by local maximums had I not continued the search.

Here is a graph I generated using features { 3,5 } and the small dataset's values. Class 1 is an orange color while class 2 is the blue color. Although it isn't blatantly clear, we can see that the data sets somewhat form the concentric circle that was talked about in class, with class 1 near the center, and class 2 scattered around the outside.



**Conclusion:**

In this project we got experience working with datasets to perform classifications. We used the nearest neighbor formula in conjunction with forward selection and backward elimination to perform feature selection for various sized datasets. As we saw, forward selection and backward elimination did not always return the same result due to the way the greedy choice was made.

**Citations:**

To read in data I used the fstream documentation:
http://www.cplusplus.com/reference/fstream/fstream/

For the c++ priority queue I used:
http://www.cplusplus.com/reference/queue/priority_queue/

To help me understand how the nearest neighbor algorithm and the searches I used lecture notes