Machine Learning from Data

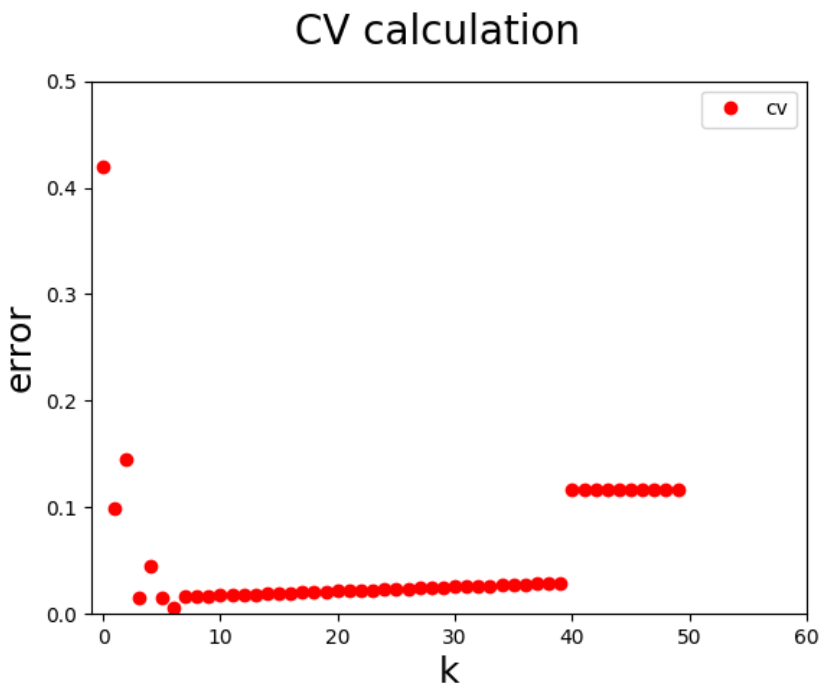
superbud9123

November 2017

k-NN Rule

a

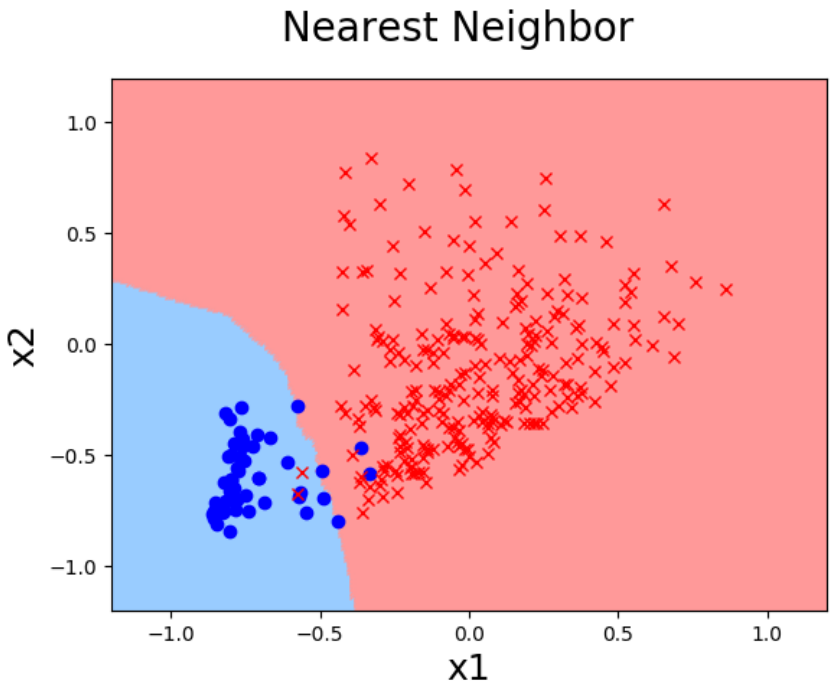
Below, I have plotted E_{cv} versus k.



The graph acts pretty similar to how I would expect. In the single digits, it creeps down to its lowest E_{cv}. Then, k gets to be too high and makes E_{cv} rise. At some point, k is out of control and causes the graph to act weirdly. I think this is because there are only a small amount of ones in this data.

I chose a value of 7 for my k.

b
Below is the decision boundary for k = 7.



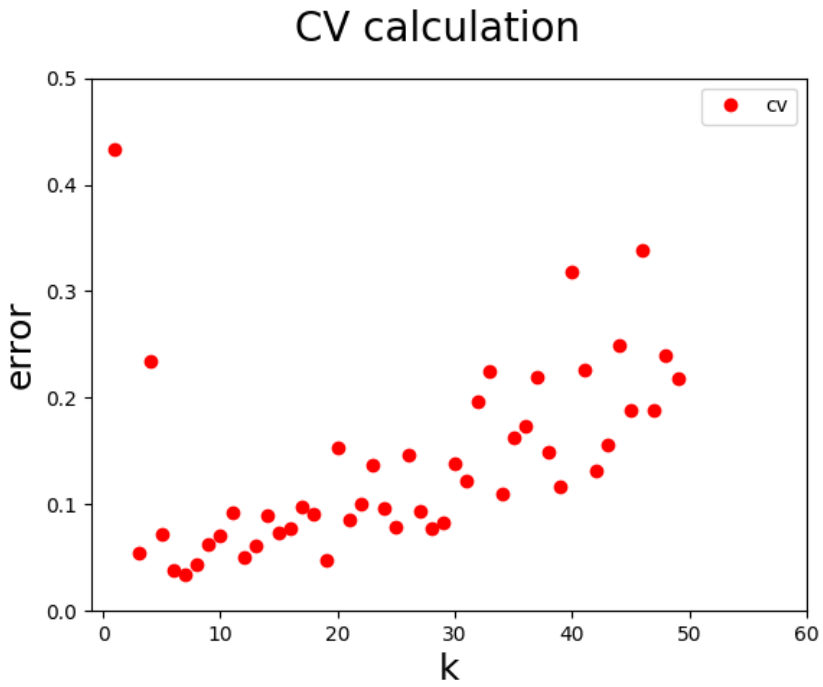
This makes sense, as the decision boundary doesn't look to be overfitted. My E_{in} was 0.031 and my E_{cv} was 0.015

c
E_{test} for 7-NN I got to be 0.030. A picture is shown below of the terminal

```
i:  9.749145934989656e-16
i:  1.0000000000000016
Etest:  0.03
drmoo@OPSYS:~/Documents/MachineLearning/HW11$ take
```

RBF-Network

a
Below, I have plotted E_{cv} versus k.

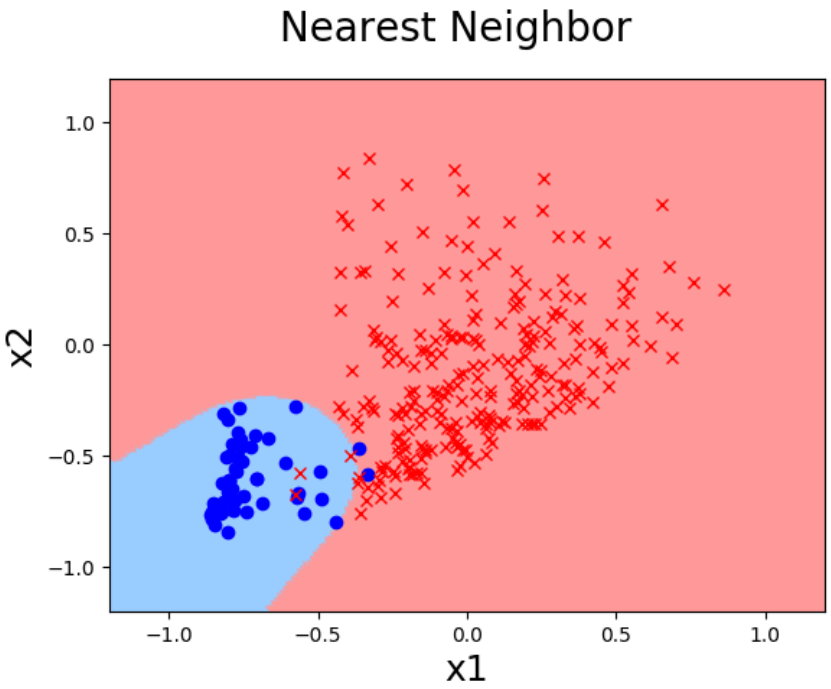


The graph acts pretty similar to how I would expect. A lower amount of beacons generally functions better. However, my result here isn't as consistent as I would like for it to be. This may also be due to the randomness of the centers generated.

I chose a value of 8 for my k.

b

Below is the decision boundary for k = 8. My E_{in} was 0.0166 and my E_{cv} was 0.032



This makes sense, as the decision boundary doesn't look to be overfitted.

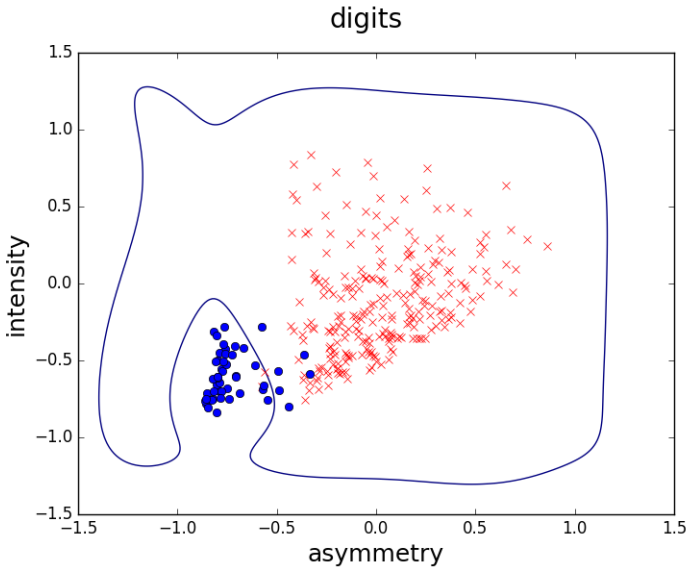
c

E_{test} for 8-NN I got to be 0.085. A picture is shown below of the terminal

```
i:  9.749145934989656e-16
i:  1.0000000000000016
Etest:  0.085
drmoo@OPSYS:~/Documents/MachineLearning/HW11$
```

Comparison

For reference, I have put the linear model with 8th order polynomial transform below.



The E_{test} for λ of 80 was 0.03178. So, below I have provided a table.

Model	E_{test}
Linear 8th Order	0.03178
k-NN	0.030
k-RBF	0.085

In general, this result makes sense. The features that we have assigned to this data separate the data very well. As a result, the models we have selected perform well on our data.

The k-NN neighbor performs the best, and I believe this makes sense as well. The similarity of our data is definitely noticeable. After looking at our data, it is clear that there are very few outliers. As a result, we only need 7-NN to essentially erase the output effect of those outliers.

Even though k-NN performs the best, it is also good to note that all of these models give an answer within a small range. This makes sense because these are all proven and effective methods. Once we have regularized them using cross validation, we can assume our output is pretty close to an ideal f(x).

It is also good to keep in mind that data snooping has occurred at all steps of this process. A few homeworks ago, we generated our test and training points and normalized them all at the same time. We are still using that data, so all of this is with very small data snooping