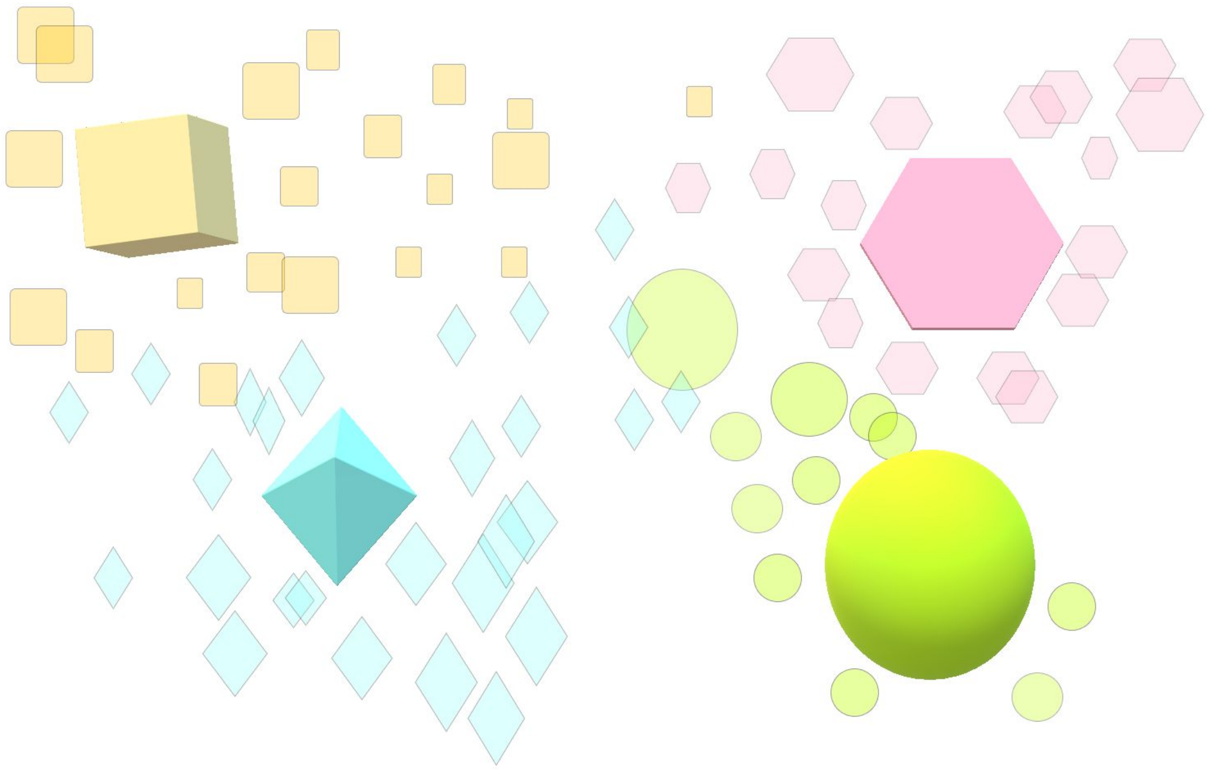
PROJECT REPORT ASSIGNMENT 2

Introduction to Machine Learning and Data Mining

Spring 2018



Lino Valdovinos

861300001

CS-171

INTRODUCTION

For this assignment I had the task of developing a K-Nearest Neighbor Classifier using Cross Validation with 10% folds to test the accuracy of the data. In addition to the KNN Classifier, for the extra credit portion of this assignment I implemented a perceptron classifier which works really well.

PROCEDURE

KNN:

LP-Norm distance, p set to 2, is used to the K nearest neighbors where k ranges from [1-10]. Then based of the class of the nearest neighbor the instance is then classified. In addition, I shuffle the data randomly and then performed cross validation by selecting 10% of the data to be tested and 90% percent to be training data. After I get our prediction for the for the 10% of the data I compare the predicted classes with the actual ones and get an calculate the error rate, accuracy, sensitivity, and specificity. Then proceed the next 10% of the data as testing and the all the other data including the one I previously tested with as training data. The Cross validation repeated until it is done testing all the data points. I repeat this process with different k values ranging from [1-10], as well as different p values of 1 and 2.

Perceptron:

The perceptron involves adding and extra feature to the data the is equal to 1 to all data points. Then I multiply all the features with a set of respective weights, weights are either 0 or a small random number between [-1,1],and add those results up together. I past the results through and activation function and return our prediction for the point. In this case the activation function is the sign(a) function which return 1 if a > 0 and otherwise it returns -1. I update all of the weights if the prediction is incorrect with the in the following way:

error = actual - sign(pred_sum)

w[k] = w[k] + error*input_x[i][k]*alpha

Where w[k] is the weight assigned to the featured defined by input_x[i][k], error is either 2 or -2, and alpha is the learning rate set to 1 for this assignment but also tested smaller values like .1. To find other local maximums. The whole process iterates through the data until the error rate is below 2.8 -2.6% then it returns the weights that are then used to classify the testing data. Cross validation is performed here as well.

DATA

The data can be found here:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)

All the missing data was handled by removing the instance from the data set. I felt that this was the best apporced since it was only 16 instances, about 2% of the data. However, a better approach would have been to replace those values with the mean of for the given class.

RESULTS

Question 1:

Results using k =1 and p = 2 with the first 80% of data points as training and the last 20% as testing.

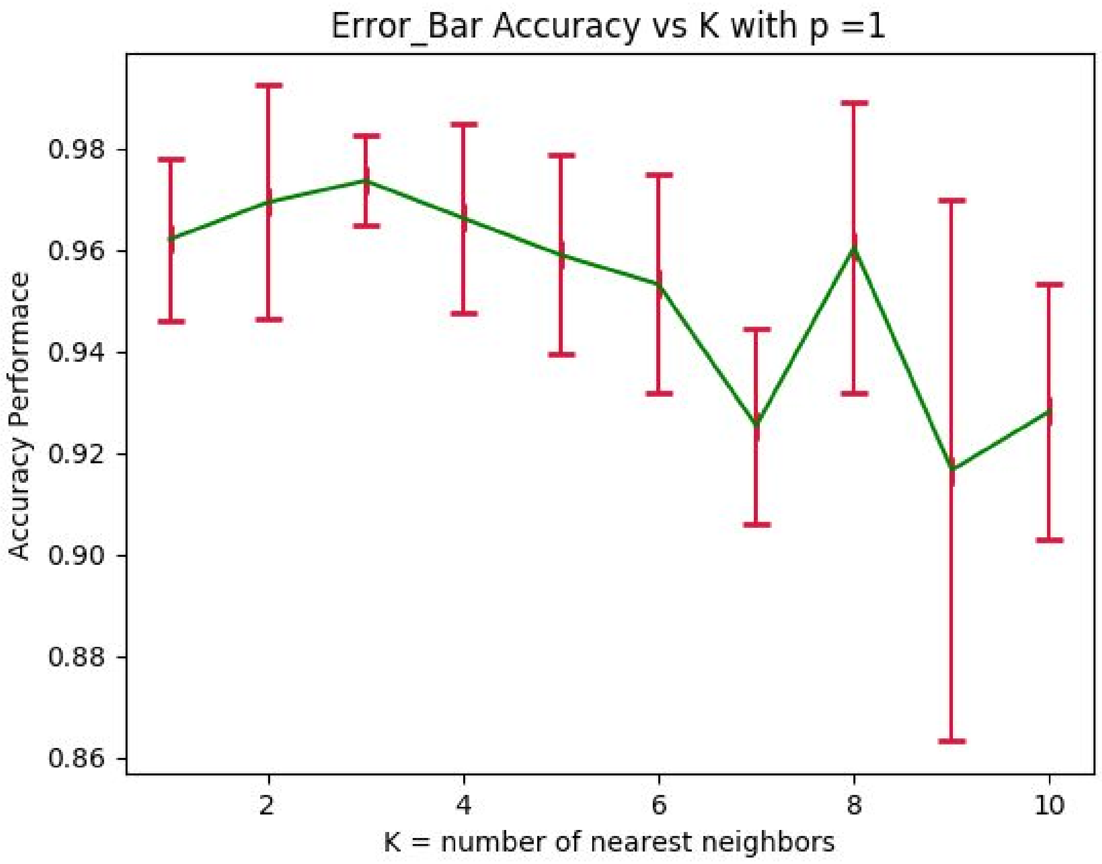
Fold range :544 - 683

Error of 0.00719424460432 (0.72 %) or Accuracy of .9928057544 (99.28%)

Question 2:

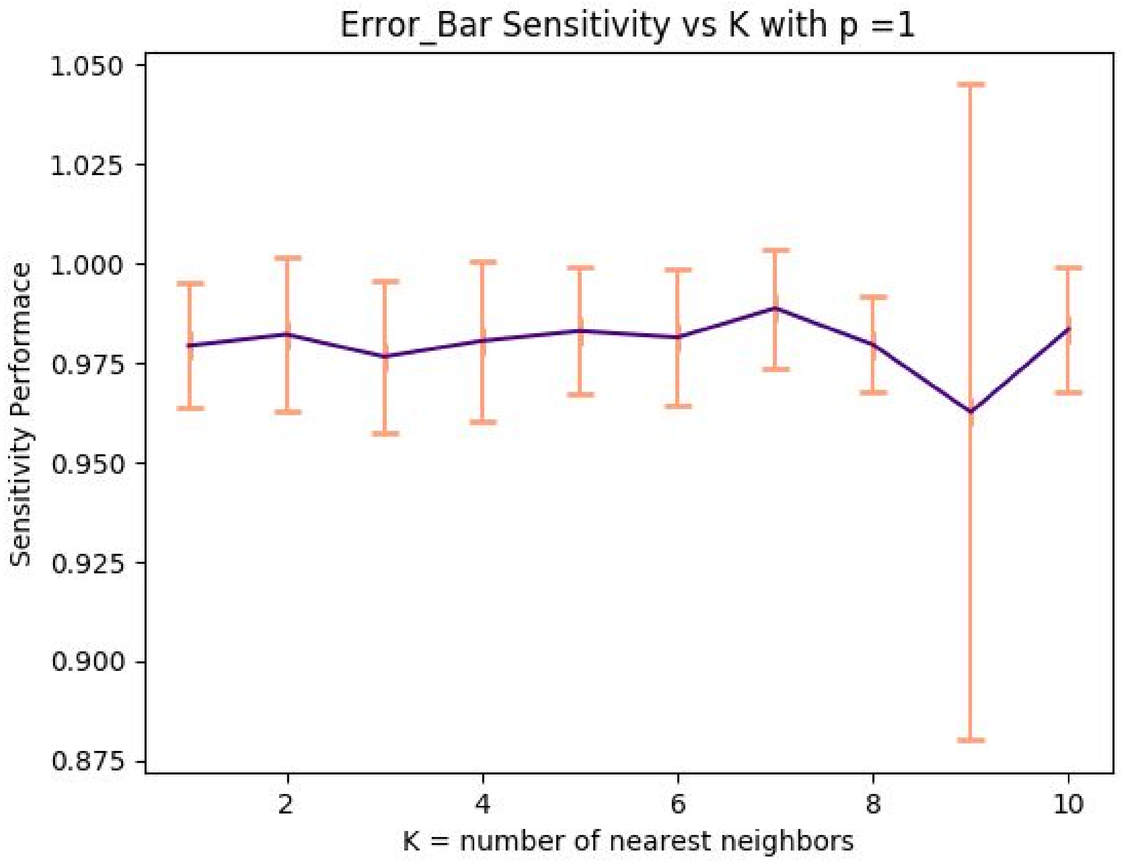
For LP_norm value p =1

For all graphs the line running horizontally represents the mean after 10 fold cross validation and the vertical lines represent the standard of deviation.



A few things to point out of the Accuracy vs K error bar with a p value equal to one.

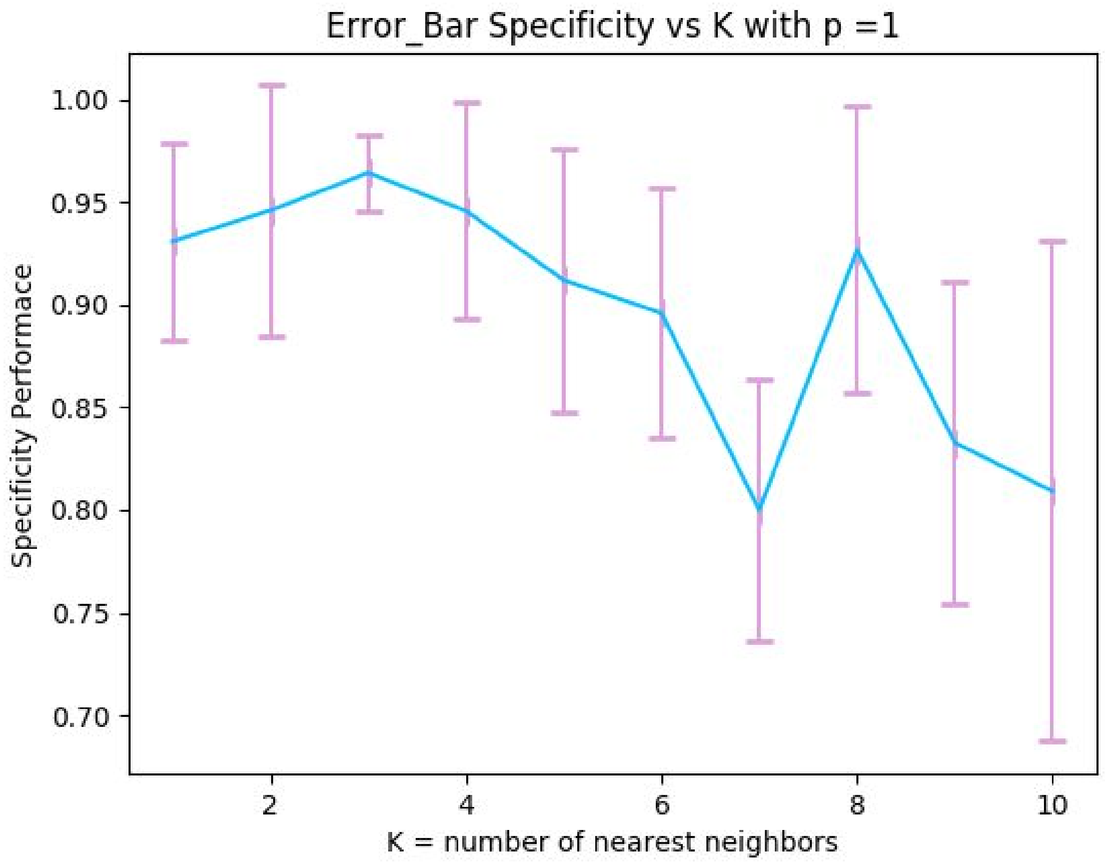
- Performance really drops down as the k values go past 3.
- K = 3 seems to be the sweet spot in this case. As accuracy stays within 1% standard of deviation of 97.
- The worse k value is 9 performs worse than 10



sensitivity: number predicted benign / total number of benign

A few things to point out of the Sensitivity vs K error bar with a p value equal to one.

- Overall we have the sensitivity is about the same, however there is a spike at 9 matching the results we got in the accuracy graph

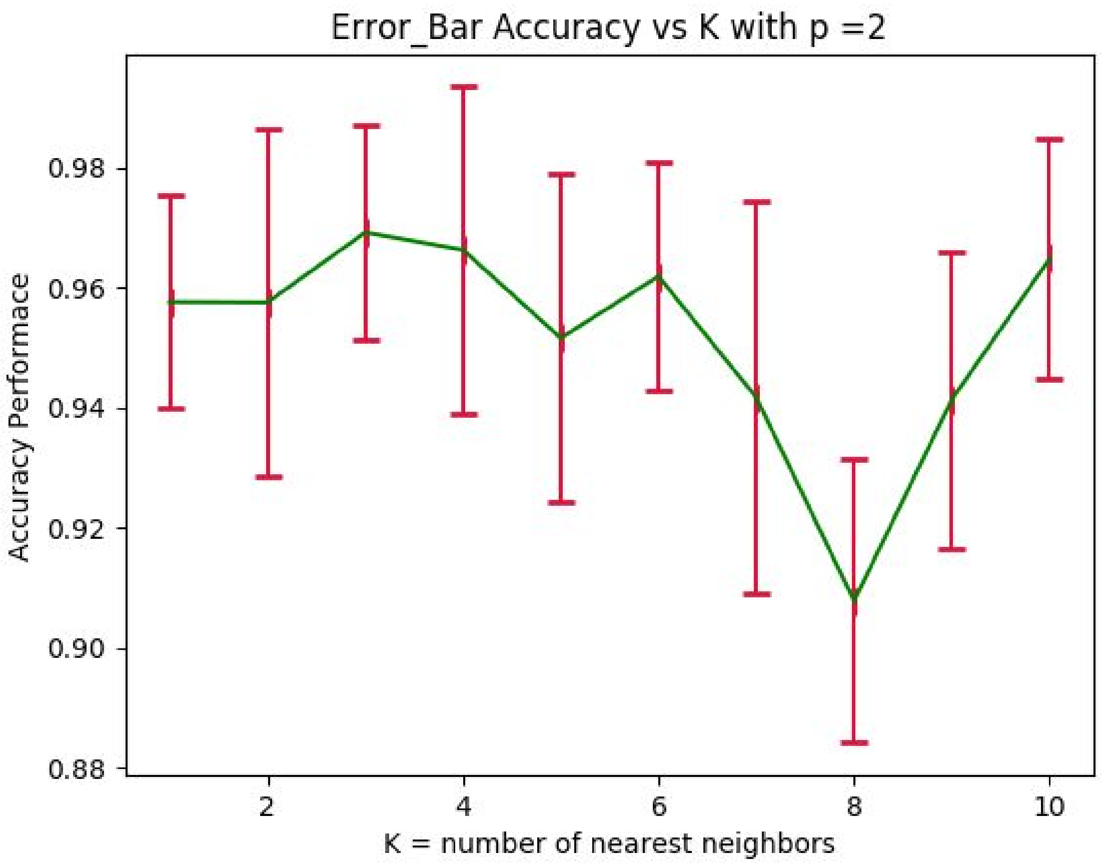


Specificity = number of predicted malignant / total number of malignant

A few things to point out of the Specificity vs K error bar with a p value equal to one.

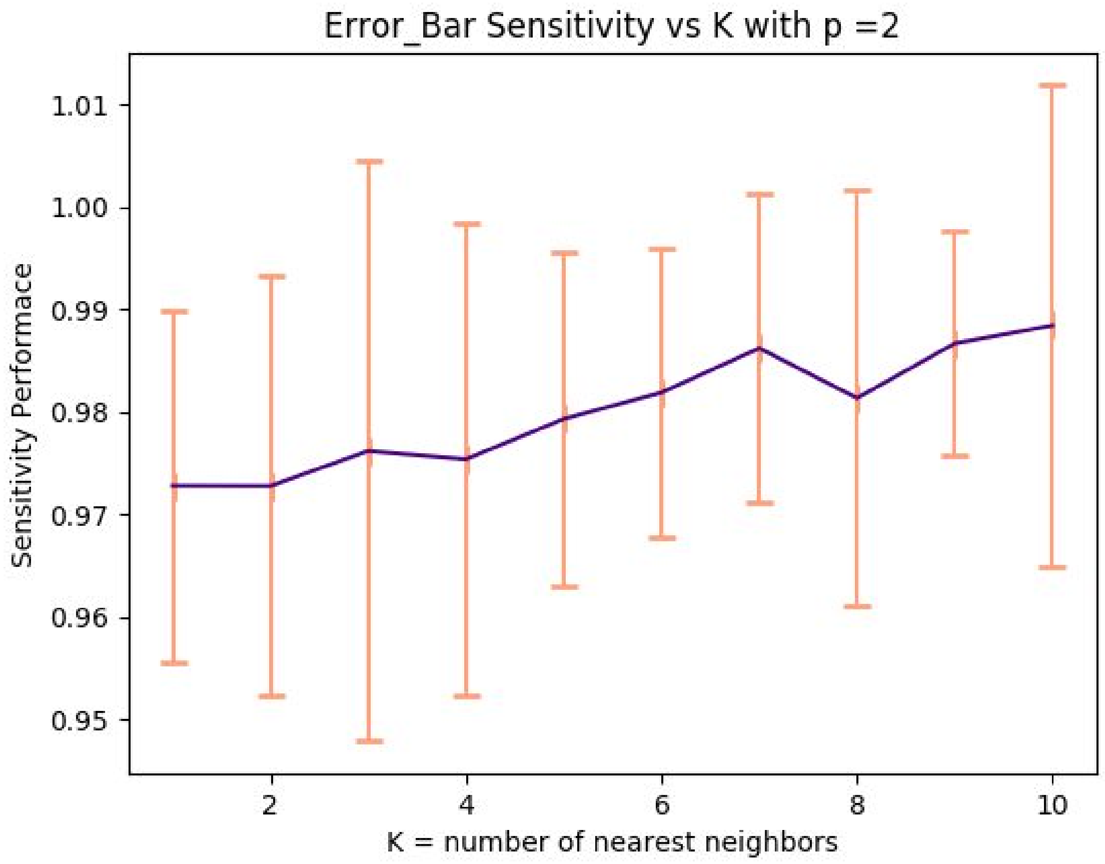
- The specificity seems to be correlated with the accuracy.
- Notice that k equal to 3 performance really well
- As K increments the performance gets worse

For LP_norm value p =2



A few things to point out of the Accuracy vs K error bar with a p value equal to two.

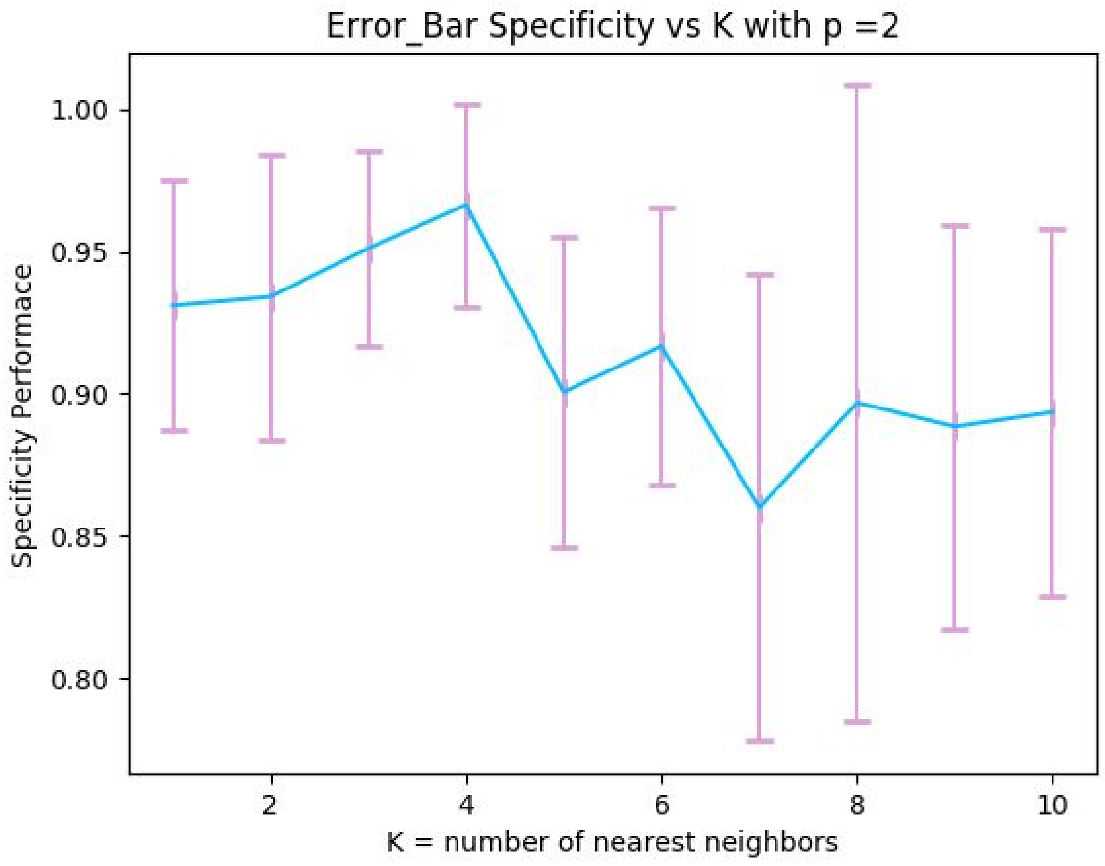
- As we notice before performance really drops down as the k values go past 3.
- Again K = 3 seems to be the sweet spot in this case. As accuracy stays within 2% standard of deviation of 97.
- In this case the worse k value is 8 performs worse than 10



sensitivity: number predicted benign / total number of benign

A few things to point out of the Sensitivity vs K error bar with a p value equal to two.

- Notice that the standard of variation for k = 3 is pretty big compared to the rest.
- 9 seems to do really well.



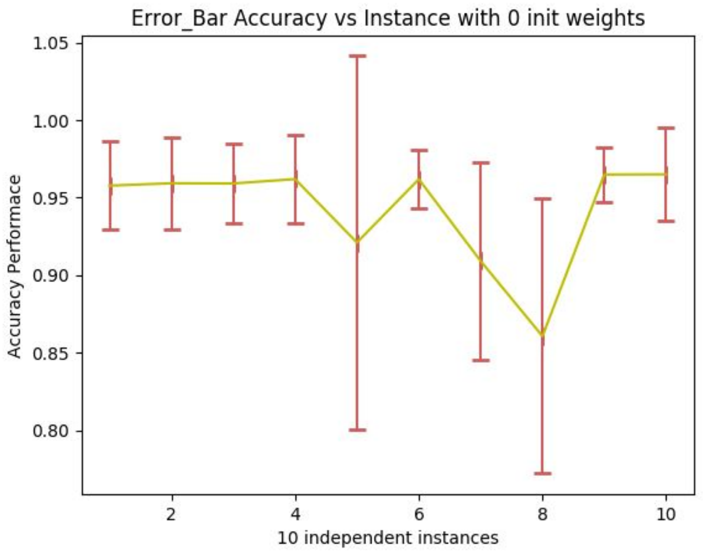
Specificity = number of predicted malignant / total number of malignant

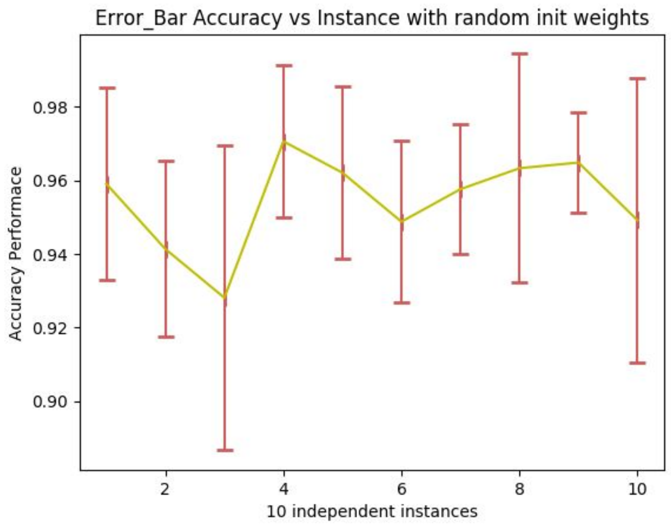
A few things to point out of the Specificity vs K error bar with a p value equal to two.

- Similarly, the specificity graph seems to be correlated with the accuracy.
- Notice that k equal to 3 performance really well but 4 seem to be better in this case
- As K increments the performance gets worse 8 also performs really bad matching the accuracy.

Question 3: Perceptron [30%] - EXTRA CREDIT

Accuracy:



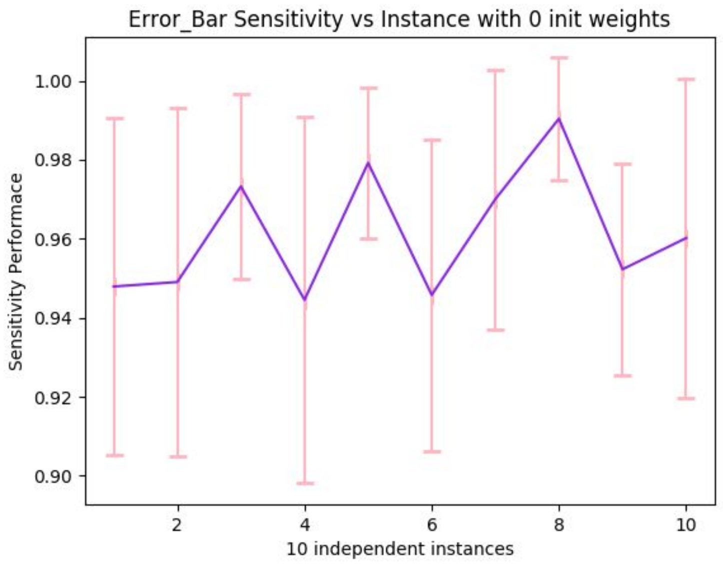


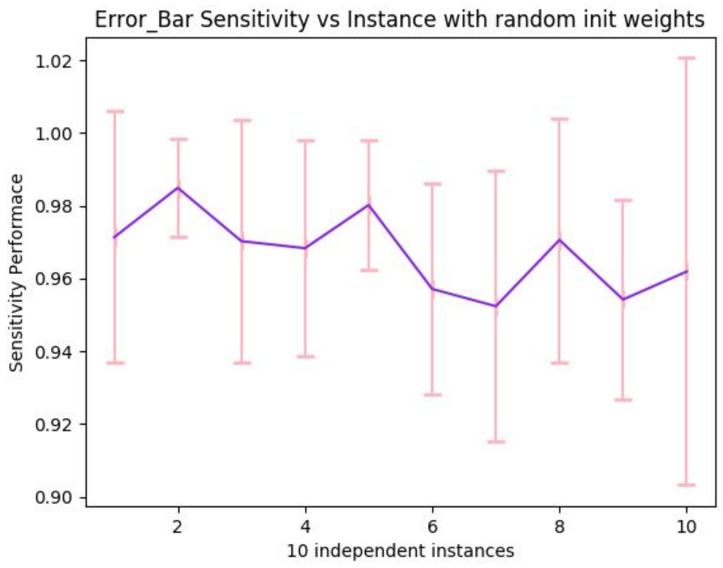
Weights initialized to 0 on the left and weights initialized to to a random number between [-1,1]. All weights were obtained only when the error rate was less the 3.6%.

Somethings to notice from the accuracy graph .

- It's hard to tell if one is automatically better to choose random weights over 0 init. But overall there is a sight performance gain in random weights.
- Number 4 and number 9 both on the right hand side seem to be the best performing instances.
- In the testing both 0 initialized weights and random weights seem to converge to similar values.

Sensitivity:





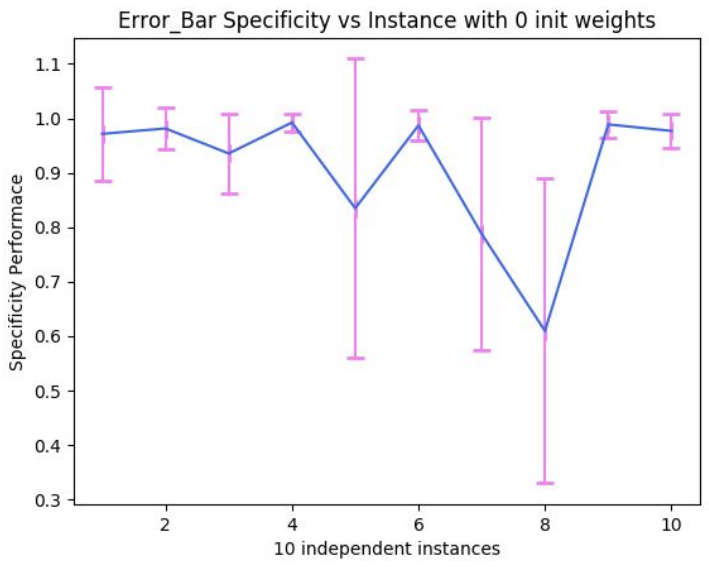
sensitivity: number predicted benign / total number of benign

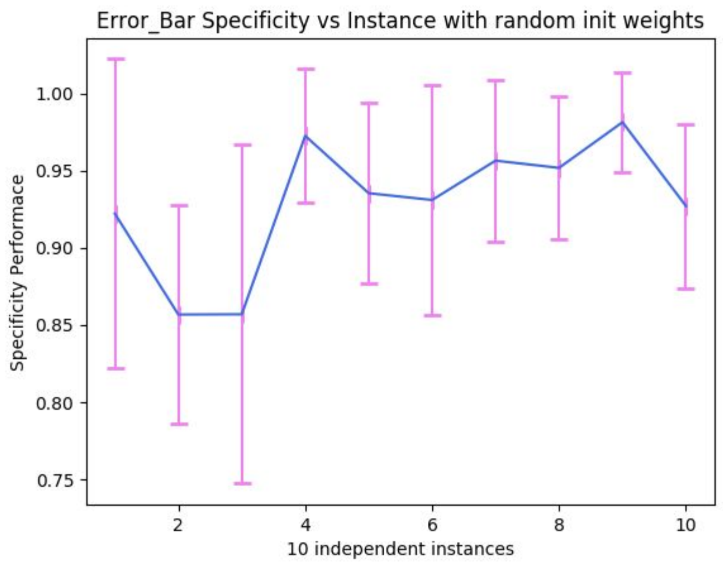
Weights initialized to 0 on the left and weights initialized to to a random number between [-1,1]. All weights were obtained only when the error rate was less the 3.6%.

Somethings to notice from the sensitivity graphs.

- It seem more clear here that random init weights have an advantage over 0 init wights. Overall the means are slightly higher while the standard of deviation seems to be more compact.
- The left graph perform poorly at labeling benign patients

Specificity:





Specificity = number of predicted malignant / total number of malignant

Weights initialized to 0 on the left and weights initialized to to a random number between [-1,1]. All weights were obtained only when the error rate was less the 3.6%.

Somethings to notice from the specificity graph .

- The left graph in a lot of instance performs really well at classifying malignant tumors however there is a lot of really bad instances as well.
- Same can be sade about the right graph with random weights but not as good.
- If classifying malignant tumors was your priority, instance number 9 on the right side probably your best choice.

CONCLUSION

To summarize the results for KNN, it seem that k value of 3 seems to be the sweet spot for both p = 1 and p =2. In addition,

There are instance of Perceptrons that outperform KNN, it just of finding a global min of at least a better performing local min. But there seems to be that potential to be the better classifier in terms of all three measures of performance. Overall, random initialized weights show improvements over zero initialized weights.

REFERENCES

Class slides

http://www.cs.ucr.edu/~epapalex/teaching/171_S18/index.html#schedule

Matplotlib documentation

https://matplotlib.org/index.html

Numpy documentation

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.array.html

Python documentation

https://docs.python.org/2/library