William Briguglio

# Assignment 2:

1) Following are the results of running an SVM implemented with scikit on the four provided datasets. A linear, polynomial(d=2), and RBF kernel were tested on each dataset.

```
Data Set: twospirals.dat
      Support Vector Machine(Kernel = linear):
            PPV: 0.65053  NPV: 0.63619  Specificity: 0.66800
      Sensitivity: 0.61800  Accuracy: 0.64300
Support Vector Machine(Kernel = poly):
            PPV: 0.61966  NPV: 0.53655  Specificity: 0.82200
      Sensitivity: 0.29000  Accuracy: 0.55600
Support Vector Machine(Kernel = rbf):
            PPV: 0.95219  NPV: 0.95582  Specificity: 0.95200
            Sensitivity: 0.95600  Accuracy: 0.95400

Data Set: twogaussians.dat
      Support Vector Machine(Kernel = linear):
            PPV: 0.97087  NPV: 0.96907  Specificity: 0.96907
      Sensitivity: 0.97087  Accuracy: 0.97000
Support Vector Machine(Kernel = poly):
            PPV: 0.97073  NPV: 0.96410  Specificity: 0.96907
      Sensitivity: 0.96602  Accuracy: 0.96750
Support Vector Machine(Kernel = rbf):
            PPV: 0.98522  NPV: 0.96954  Specificity: 0.98454
            Sensitivity: 0.97087  Accuracy: 0.97750

Data Set: clusterincluster.dat
      Support Vector Machine(Kernel = linear):
            PPV: 0.41657  NPV: 0.00000  Specificity: 0.00000
      Sensitivity: 0.71400  Accuracy: 0.35700
Support Vector Machine(Kernel = poly):
            PPV: 1.00000  NPV: 1.00000  Specificity: 1.00000
      Sensitivity: 1.00000  Accuracy: 1.00000
Support Vector Machine(Kernel = rbf):
            PPV: 0.99602  NPV: 1.00000  Specificity: 0.99600
            Sensitivity: 1.00000  Accuracy: 0.99800

Data Set: halfkernel.dat
      Support Vector Machine(Kernel = linear):
            PPV: 0.68346  NPV: 0.81918  Specificity: 0.59800
      Sensitivity: 0.86800  Accuracy: 0.73300
Support Vector Machine(Kernel = poly):
			PPV: 0.70721  NPV: 1.00000  Specificity: 0.58600
		Sensitivity: 1.00000  Accuracy: 0.79300
	Support Vector Machine(Kernel = rbf):
			PPV: 1.00000  NPV: 0.99800  Specificity: 1.00000
			Sensitivity: 0.99800  Accuracy: 0.99900
```

2) Performance Notes:

Two Spirals dataset:

For this dataset, as expected, the linear classifier preformed poorly with 64% accuracy however the polynomial preformed even worse with a 56% accuracy. This is because the data is not linearly separable so the linear preformed poorly, and additionally in the polynomial space, the data must not be linear separable either. However, perhaps adding more features with a degree 3 polynomial kernel would improve the classification. Finally, the RBF kernel preformed exceptionally well, with an accuracy of 95%. This is because the RBF kernel effectively measures the Euclidian distance between two points, so here, it acts similar to the way a k-NN neighbour classifier would act and since the classes are in two distinct regions, judging a sample's class by its nearest neighbours is an effective classification strategy.

Two Gaussians dataset:

For this dataset all three preformed equally as well with an accuracy of $97\% \pm 0.75\%$. This is because the data is already linearly separable in the data's initial feature space, so the linear kernel works well and the other two are just over kill.

Cluster in Cluster dataset:

For this dataset, the linear classifier preformed poorly with 35% accuracy(worse then random) However both the polynomial and RBF kernel achieved a $99.9\% \pm 0.01\%$ accuracy. This is because the polynomial features added by the polynomial function were able to be linearly separated in the new feature space. Additionally, the RBF kernel was effective due to the data lying in distinct regions.

Half Kernel dataset:

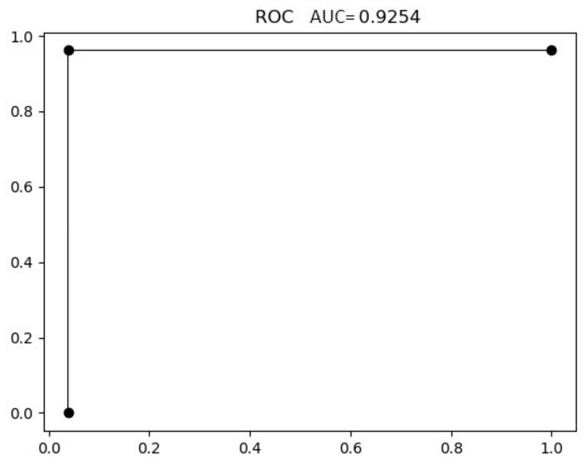
For this dataset the linear classifier preformed better then expected with a 73% accuracy however upon inspection of the plotted classified data, it is clear the SVM just found the optimal place to draw a straight line for best classification results. The polynomial preformed mildly well with a 79% accuracy but once again, inspecting the plotted results shows the classifier is misclassifying a large region and perhaps a degree 3 kernel would perform better. Finally, the RBF kernel preformed exceptionally well with a 99.9% accuracy. This performance is due to the same reason as before, the fact that the data is in two distinct regions.

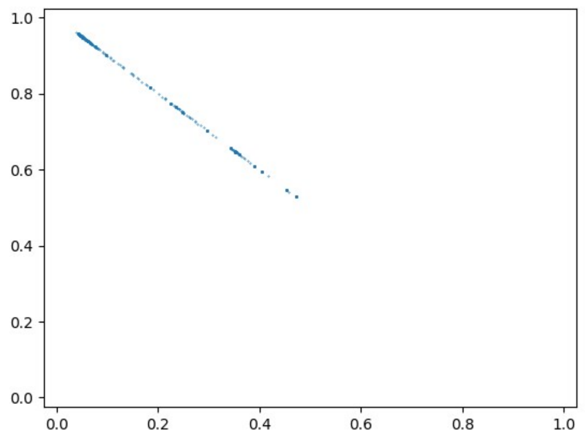
3) Following are the ROC curves obtained for the RBF classifier on each dataset, along with the points that were used to construct the ROC curve. The x-axis and y-axis are extended by 0.025 as some points would be lying directly on the edges of the graph if the two axes went from 0 to 1.

Note: each of the datasets had a parameter(C, $\gamma\gamma$) pair that resulted in a classifier with both minimum false positive rate and maximum true positive rate, meaning that when the convex hull was taken to remove the sub-optimal classifier points, only one remained, creating a square ROC curve. i.e.) Because all the removed points all had more false positives and less true positives then the point that remained

in the convex hull, regardless of which efficiency measure we prioritize, the same (C, $\gamma\gamma$) pair remained the "best", so a single (C, $\gamma\gamma$) pair defined the shape of the ROC curve.
Note II: Each Data set was tested with all combinations of C and $\gamma\gamma$ values where $C = 2^{ii}$ and $\gamma\gamma = 2^{jj}$ such that $ii \in [0,20]$ and $jj \in [0,8]$ giving a total of 189 points, except for the Two Spirals dataset, which was tested with $ii \in [-20,20]$ and $jj \in [-8,8]$, yielding 697 points, as a totally optimal solution was not found in the initial smaller range.

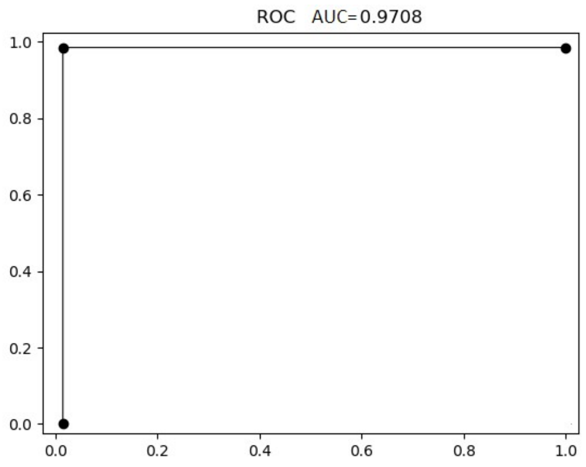
Two Spirals dataset:

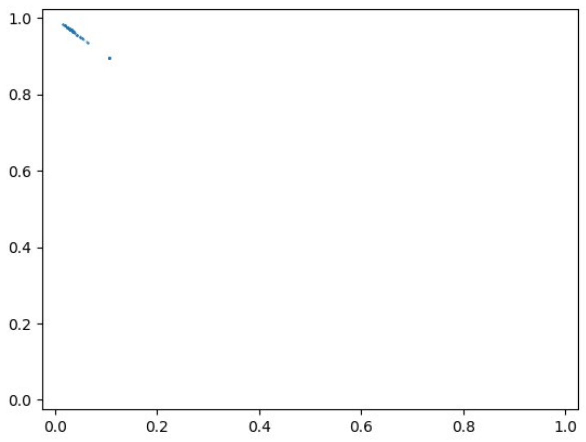




Here, the SVM's efficiency was a lot more dependent on its parameter's values with some parameters as bad a random classifier and the best parameters being $C = 2^{-2}$ and $G = 2^{1}$ achieving a TRP of 96.2% and an FPR of 3.8%

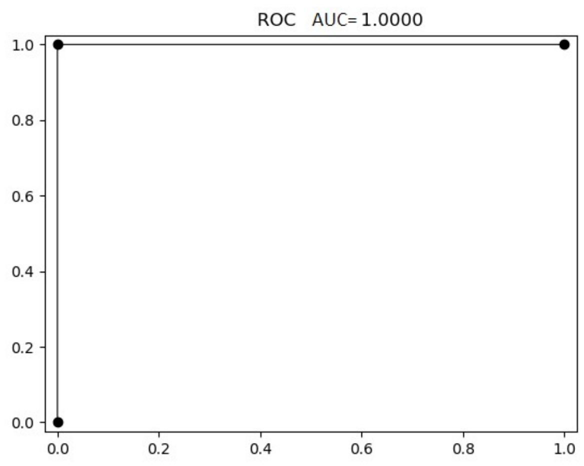
Two Gaussians dataset:

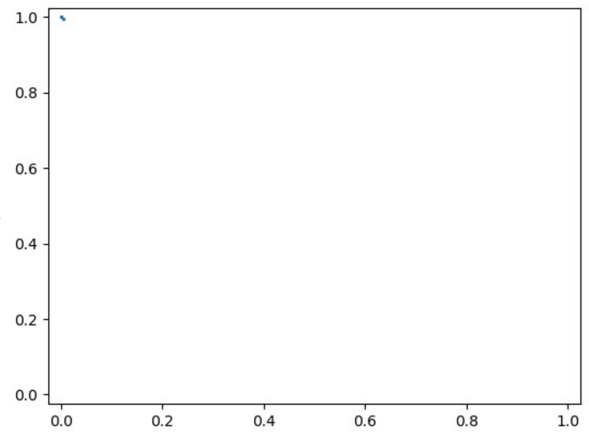




Here, all 189 points are above 90% TPR and below 15% FPR

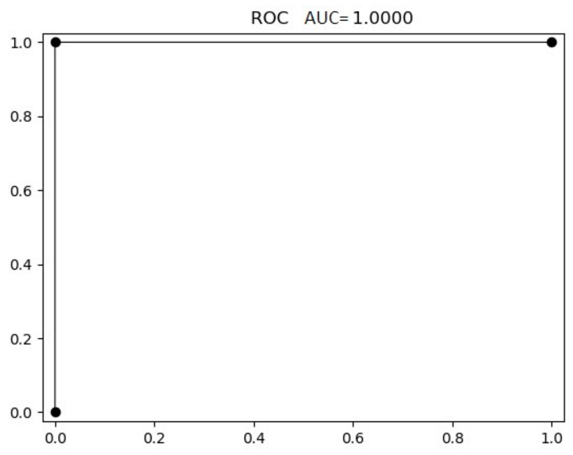
Cluster in Cluster dataset:

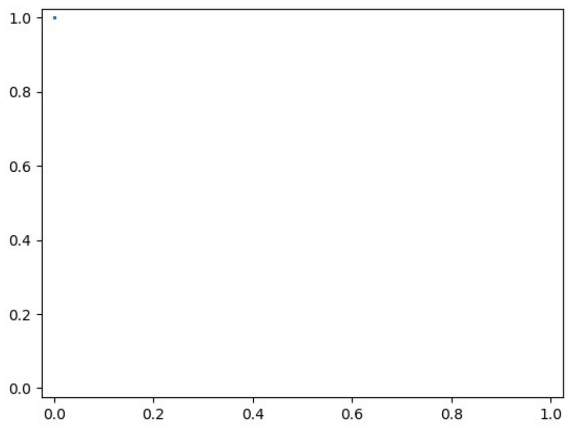




Here, all 189 points are at or near 100% TPR and 0% FPR

Half Kernel dataset:



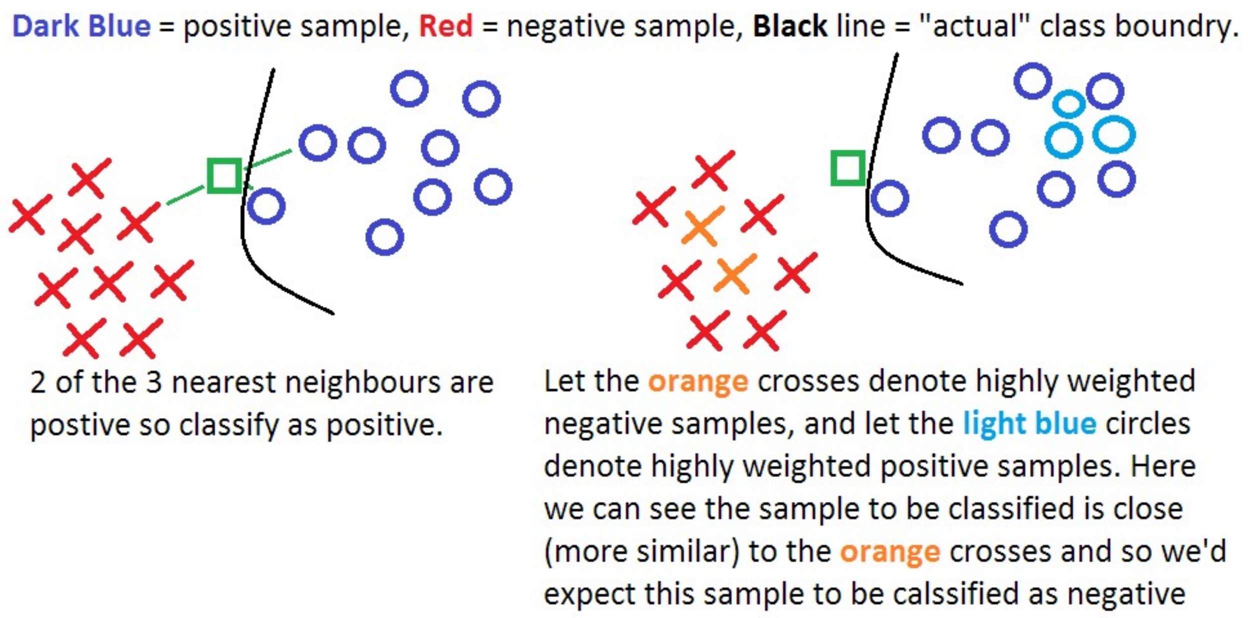


Here also, all 189 points are at or near 100% TPR and 0% FPR

4) The SVM-R was able to improve upon the K-NN neighbours by 1-4% accuracy depending on the data set. This similar performance is due to the fact that k-NN neighbours and RBF kernel both use a similarity function(Euclidian distance) between two samples as features.

The RBF kernel with Euclidian distance highlighted in red: $KKxx_{ii}, xx_{jj} = \exp \frac{xx^{ii,xxjj}\ ^{2}}{2\sigma\sigma 2}$

Since the data lies in two distinct regions, both classifiers were able to accurately classify the data. The increase in performance, however small, could be due to the RBF's ability to weight certain neighbours as less important, so a sample on the edge of the two regions, that is very close to both positive and negative samples would still be closer to the samples lying in the centre of one region then the samples lying in the centre of another and thus can be more accurately classified if we ignored the samples that are near but also lying on the edge of the regions. This is illustrated below.



Dark Blue = positive sample, Red = negative sample, Black line = "actual" class boundry.

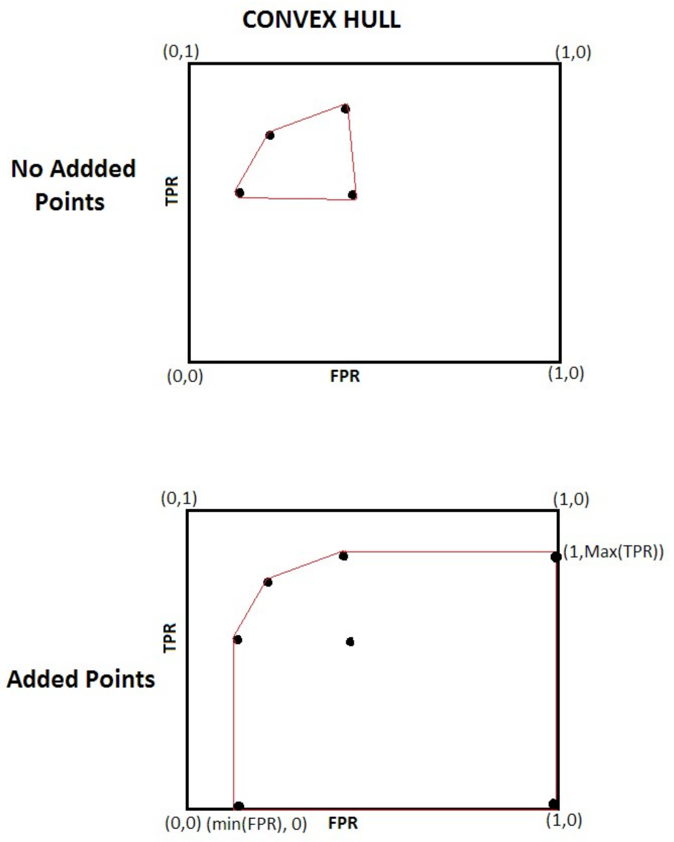
2 of the 3 nearest neighbours are postive so classify as positive.

Let the orange crosses denote highly weighted negative samples, and let the light blue circles denote highly weighted positive samples. Here we can see the sample to be classified is close (more similar) to the orange crosses and so we'd expect this sample to be calssified as negative

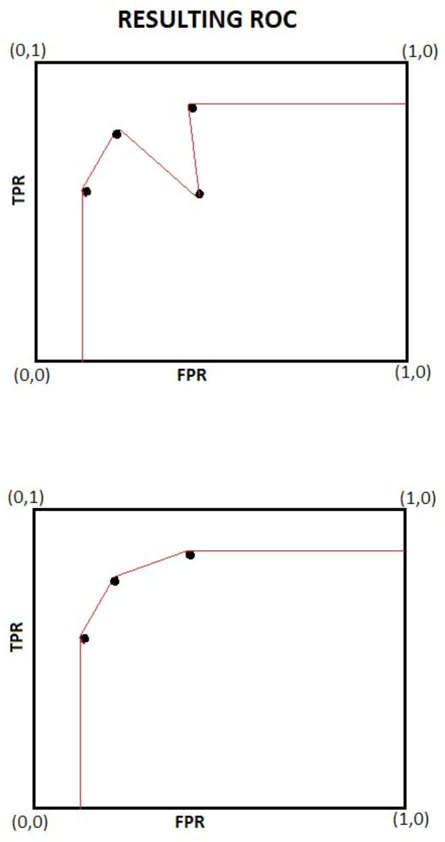
## Implementation Notes:

The SVM's were implemented using scikit's sklearn library. For implementing the ROC curve, a self implemented solution was used rather then using a library. The ROC curve was obtained by first running the SVM-R with every combination of parameter values in a certain range and saving their respective true positive and false positive rates. Next three points were added to the set of (TPR, FPR) pairs to be plotted; (min(FPR), 0.0), (1.0, max(TPR)), and (1,0). To illustrate the reason for adding these points, refer to the image below. Next, the convex hull was found using scipy's `ConvexHull()` function and the points in the hull were plotted to obtain the ROC. To obtain the AUC,  numpy's `trapz()` function was used which uses the composite trapezoidal rule to calculate the area under a set of points with straight lines drawn between them. This is the same method the sklearn's `auc()` function uses.

William Briguglio



CONVEX HULL
RESULTING ROC
No Addded Points
Added Points
(0,1)
(1,0)
(0,0)
TPR
FPR
(1,Max(TPR))
(min(FPR), 0)



(0,1)
(1,0)
(0,0)
TPR
FPR