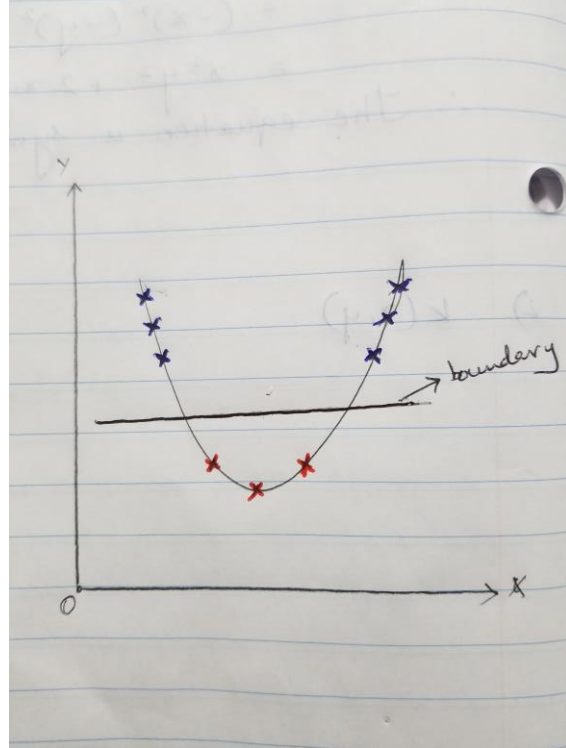


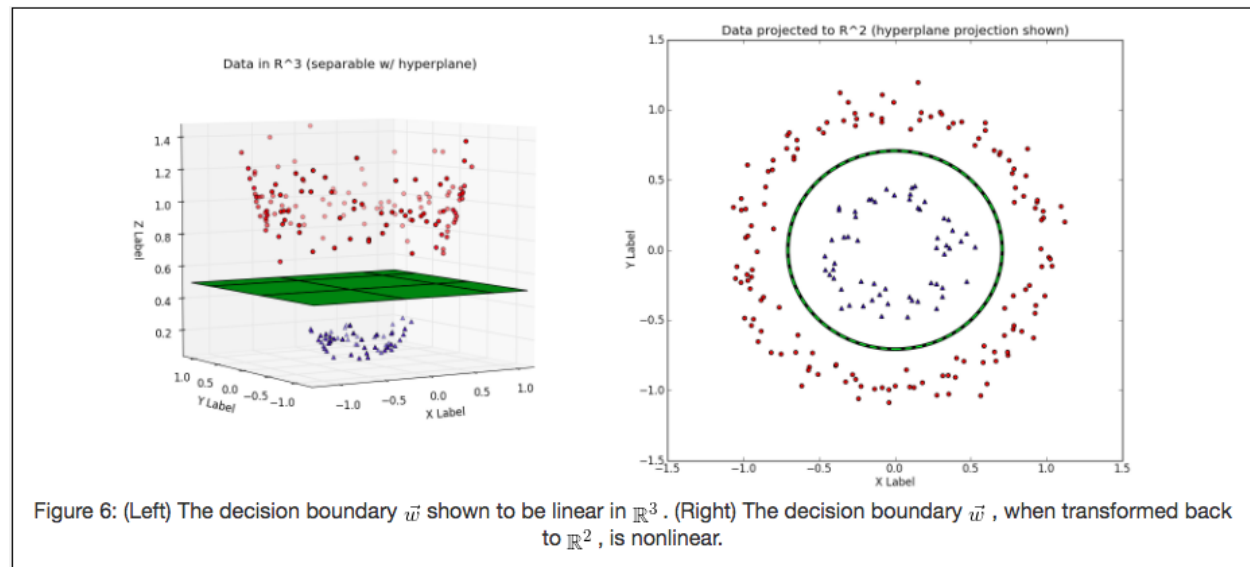
Homework -3  
Machine Learning – Fall 2017  
University Of Houston

Priscilla Imandi  
PS ID : 1619570

### Question – 1



### Question – 2,3



The equation of the plane is of the form :  $aX_1 + bX_2 + cX_3 = d$ , Here  $X_3$  can be the new dimension.

The separation in  $\mathbb{R}^2$  is :  $X_1^2 + X_2^2 = R^2$ , where  $R$  is the distance from the centre of the concentric circles to the separation boundary.

A linear classifier draws a non linear boundary because the data is projected into higher dimensions. If a curve(non-linear) is necessary to separate data points in a lower dimension (2D), then the input points can be projected into a higher dimension (e.g 3D) such that a plane(linear) can separate those points.

Input space is the space with all the input variables in the lower dimension. These are mapped to the higher dimension from which features are extracted. This higher dimensional space is called the feature space.

#### Question – 4

If an algorithm is described solely in terms of inner products in input space then it can be lifted into feature space by replacing occurrences of those inner products by  $k(x, x')$ ; this is sometimes called the kernel trick.

Replacing the inner products  $\langle x, x' \rangle$  in an algorithm with a kernel  $k(x, x')$  is called “kernelizing” the algorithm. That’s all there is to the kernel trick.

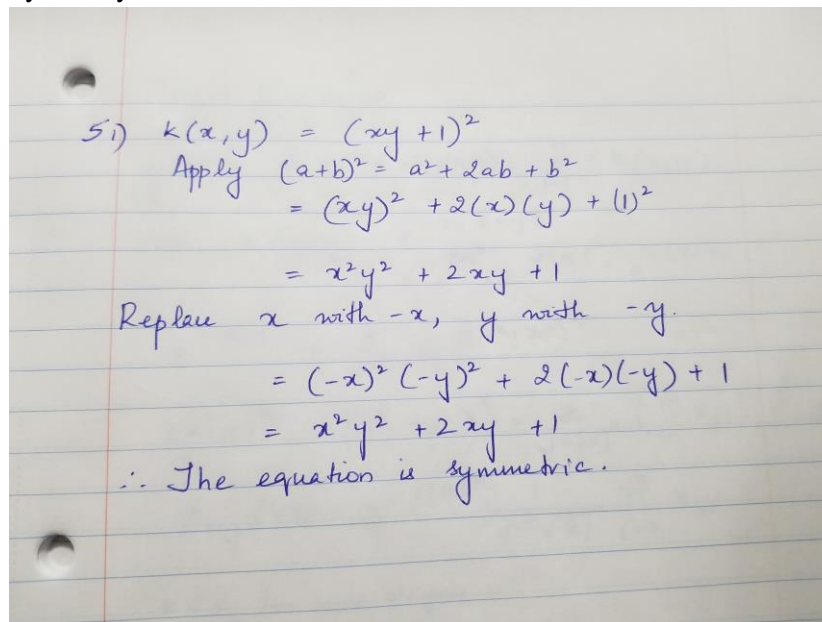
For a dataset with  $n$  features ( $\sim n$ -dimensional), SVMs find an  $n-1$  dimensional hyperplane to separate it (let us say for classification). Thus, SVMs perform very badly with datasets that are not linearly separable. It’s possible to transform our not-linearly-separable dataset into a higher-dimensional dataset where it becomes linearly separable, but, the number of dimensions you have to add (via transformations) depends on the number of dimensions you already have (and not linearly) For datasets with a lot of features, it becomes next to impossible to try out all the interesting transformations. This is where we use Kernel trick.

For a given pair of vectors (in a lower-dimensional feature space) and a transformation into a higher-dimensional space, there exists a function (The Kernel Function) which can compute the dot product in the higher-dimensional space without explicitly transforming the vectors into the higher-dimensional space first.

#### Question – 5

i  $k(x, y) = (xy + 1)^2$

Symmetry:



5)  $k(x, y) = (xy + 1)^2$   
Apply  $(a+b)^2 = a^2 + 2ab + b^2$   
 $= (xy)^2 + 2(x)(y) + (1)^2$   
 $= x^2y^2 + 2xy + 1$   
Replace  $x$  with  $-x$ ,  $y$  with  $-y$ .  
 $= (-x)^2(-y)^2 + 2(-x)(-y) + 1$   
 $= x^2y^2 + 2xy + 1$   
 $\therefore$  The equation is symmetric.

Positive Definite:

The inner product should be non-zero.

And eigen values should be positive.

For the above equation, eigen values :

i)  $x = -1, y = -1$

ii)  $x = 1, y = 1$

So, it is a valid kernel.

ii  $k(x, y) = (xy - 1)^3$

Symmetry:

5.2)  $k(x, y) = (xy - 1)^3$   
 Apply  $(a-b)^3 = a^3 - 3a^2b + 3ab^2 - b^3$   
 $= (xy)^3 - 3(xy)^2(1) + 3(1)(xy)^2 - (1)^3$   
 $= x^3y^3 - 3x^2y^2 + 3xy - 1$   
 Replace  $x$  with  $-x$ ,  $y$  with  $-y$   
 $= (-x)^3(-y)^3 - 3(-x)^2(-y)^2 + 3(-x)(-y) - 1$   
 $= x^3y^3 - 3x^2y^2 + 3xy - 1$   
 $\therefore$  The equation is symmetric.

Positive Definite:

The inner product should be non-zero.

And eigen values should be positive.

For the above equation, eigen values :

iii)  $x = -1, y = 1$

iv)  $x = 1, y = -1$

So, the equation is invalid kernel.

## Question - 6

Taylor Expansion

$$f(x) = \sum_{i=1}^m \alpha_i (\phi(x), \phi(y)) = \sum_{i=1}^m \alpha_i k(x, y)$$

Now,  $k(x, y) = e^{-\gamma \|x - y\|^2}$

$$= e^{-\gamma \|x\|^2} \cdot e^{-\gamma \|y\|^2} \cdot e^{\gamma \langle x, y \rangle}$$

$$e^{\gamma \langle x, y \rangle} = \sum_{k=0}^{\infty} \frac{1}{k!} (\gamma \langle x, y \rangle)^k$$

$$\langle x, y \rangle^k = \left( \sum_{i=1}^d x_i y_i \right)^k = \sum_{j \in [d]^k} \left( \prod_{i=1}^k x_{j_i} \right) \left( \prod_{i=1}^k y_{j_i} \right)$$

$$\phi_{k,j}(x) = e^{-\gamma \|x\|^2} \cdot \frac{1}{\sigma^k \sqrt{k!}} \prod_{i=1}^k x_{j_i}$$

$k \leq r$  for some degree  $r$ .

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

$$= e^{-\gamma \|x\|^2 - \gamma \|y\|^2} \sum_{k=0}^r \frac{1}{k!} \left( \frac{\langle x, y \rangle}{\sigma} \right)^k$$

$$|k(x, y) - \bar{k}(x, y)| \leq e^{-(\gamma \|x\|^2 + \gamma \|y\|^2)} \cdot \frac{1}{(r+1)!} \left( \frac{\langle x, y \rangle}{\sigma} \right)^{r+1} e^{\gamma \langle x, y \rangle}$$

$$\leq \frac{1}{(r+1)!} \left( \frac{\|x\| \|y\|}{\sigma^2} \right)^{r+1}$$

The dimensionality  $D$  of  $\phi$  (i.e the number of features of degree not more than  $r$ ), as presented we have  $d^k$  features of degree  $k$ . But this ignores the fact that many features are just duplicates resulting from different permutations of  $j$ . Collecting these into a single feature for each distinct

monomial (with appropriate multinomial coefficient), we have  $\binom{d+k-1}{k}$  features of degree  $k$ , and

a total if  $D = \binom{d+r}{r}$  features of degree at most  $r$ .

A simple hypothesis space (small VC-dimension) will probably not contain good approximating functions and will lead to a high training (and true) error. On the other hand a too rich hypothesis space (large VC-dimension) will lead to a small training error, but the second term in the right-hand side of (3.2) will be large. This reflects the fact that for a hypothesis space with high VC-dimension the hypothesis with low training error may just happen to fit the training data without accurately predicting new examples.

The VC-dimension  $h$  is a property of the hypothesis space  $H$  (written as  $\text{VCdim}(H) = h$ ). It is a non-negative integer which can be infinite, and is a measure of complexity or expressive power of the family of classification functions realized by the learning machine that can be implemented using  $H$ . For many cases  $h$  is proportional to the number of free parameters of  $\{f(\cdot, \alpha)\}$ . The VC-dimension is therefore a capacity measure of the family of classification functions realized by the learning machine.

VC dimension can be interpreted as the largest number of data points that can be separated in all possible ways by the functions contained in the space  $H$ . This is same as the maximal number of training examples

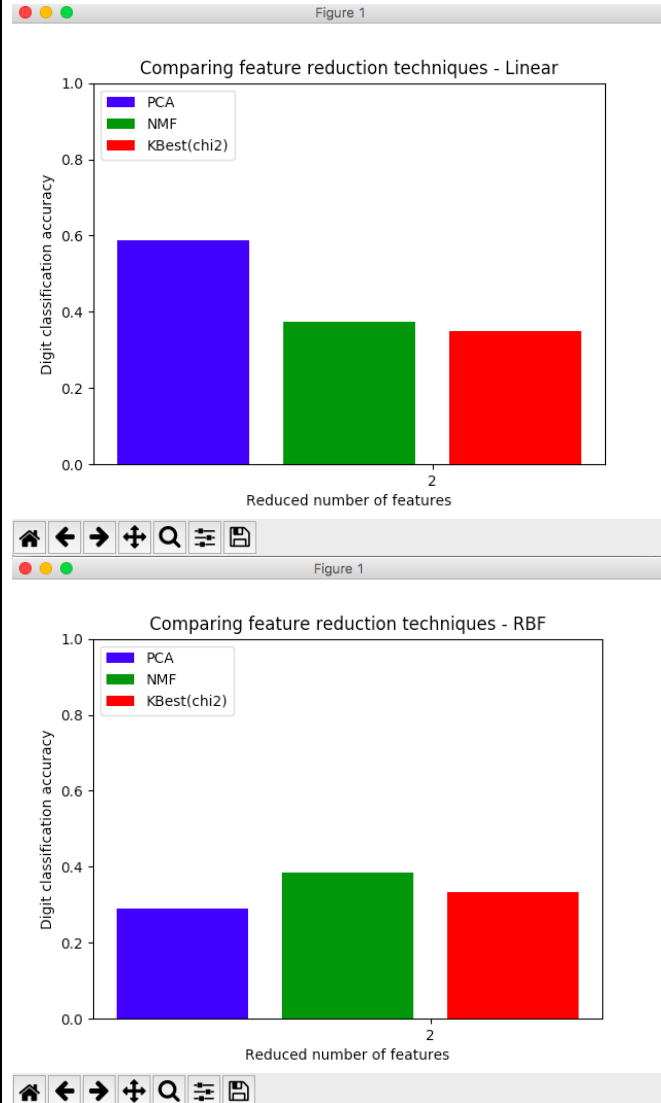
that can be learned by the machine without error for all possible binary labelings of the training data. One has to be aware that it is not necessary that for a given VC dimension  $h$ , every set of  $h$  points can be shattered. This is only guaranteed for at least one case.

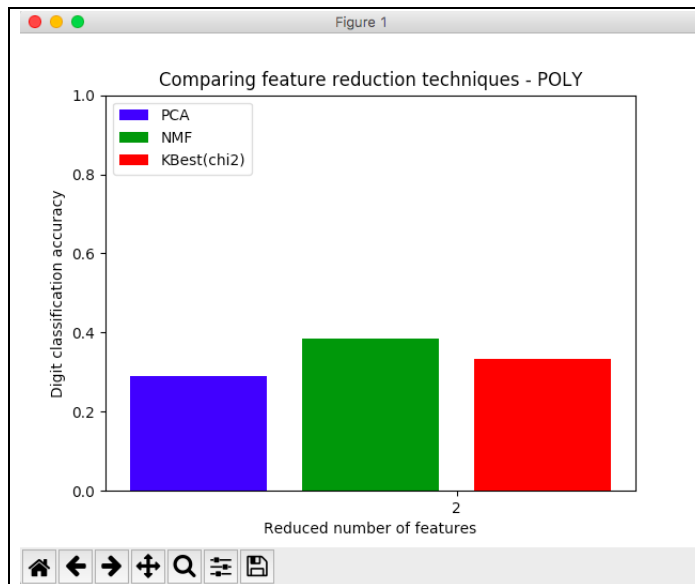
A high value of VC dimension tells us about the complexity or the capacity of the algorithm. But it is not always good to pick a high value as it might cause over fitting.

For SVM a high VC dimension is not really a problem because it has slack variables which would consider the errors that are occurring as well.

### Question 7:

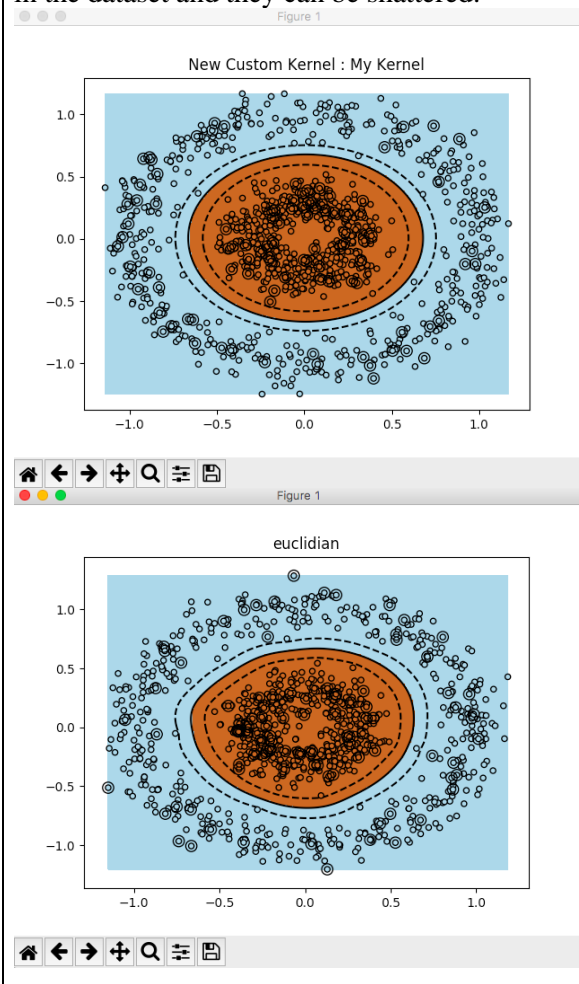
Code file included

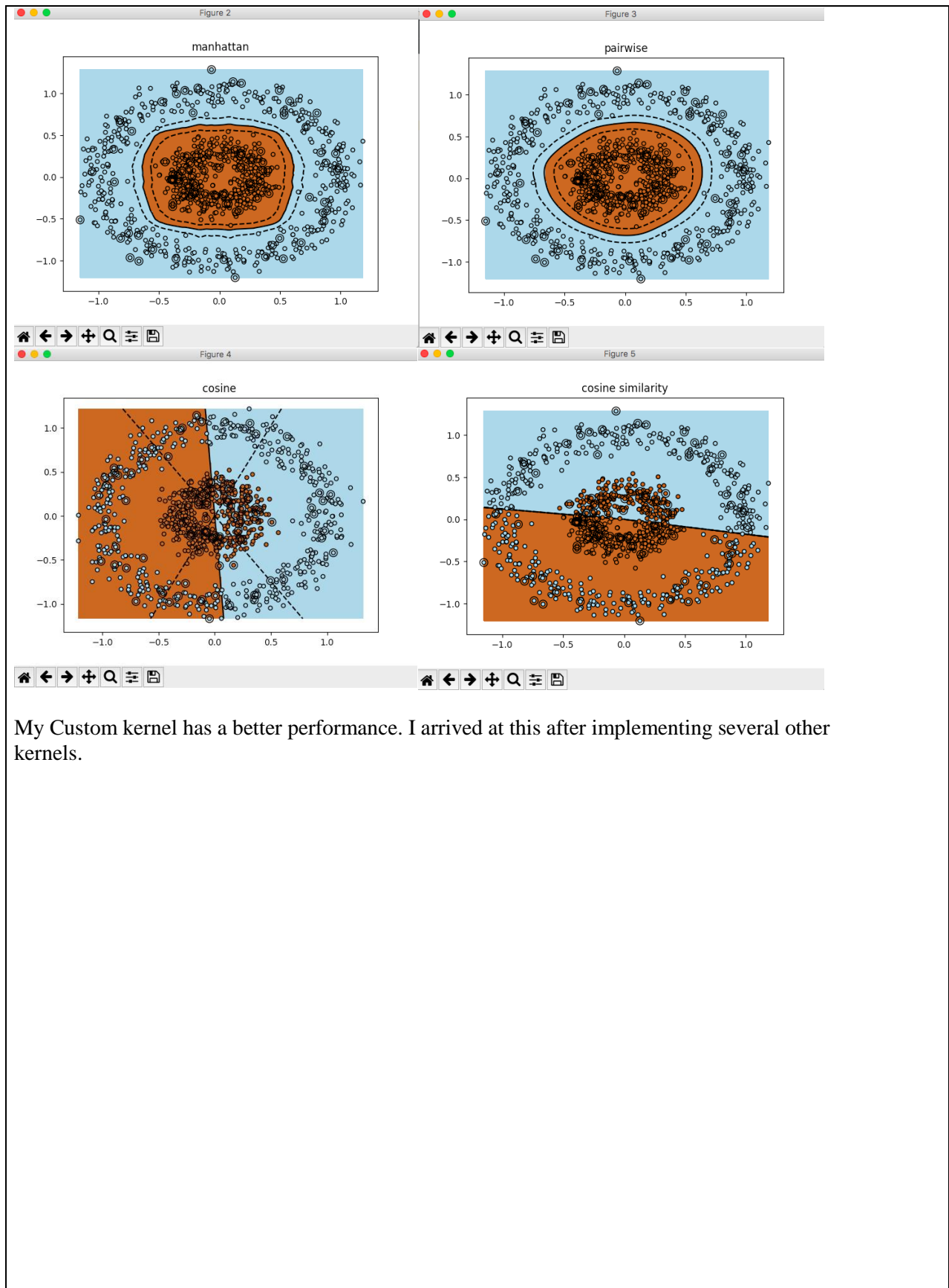




### Question 8:

The dataset is PAC learnable because the VC dimension is finite. There are a finite set of examples in the dataset and they can be shattered.





My Custom kernel has a better performance. I arrived at this after implementing several other kernels.



**Question 9:**

$$\begin{aligned}
 E[(h(x) - y)^2] &= E[h(x)^2 - 2h(x)y + y^2] \\
 &= E[h(x)^2] - 2E[h(x)]E[y] + E[y^2] \\
 &= E[(h(x) - h(x))^2] + h(x)^2 \\
 &\quad - 2h(x) \cdot f(x) + E[(y - f(x))^2] + f(x)^2 \\
 &= E[(h(x) - h(x))^2] + \\
 &\quad E[(h(x) - f(x))^2] + \\
 &\quad E[(y - f(x))^2] \\
 E[(h(x) - y)^2] &= E[(h(x) - h(x))^2] + \\
 &\quad E[(h(x) - f(x))^2] + E[(y - f(x))^2] \\
 &= \text{Var}(h(x)) + \text{bias}(h(x))^2 + E[\epsilon^2] \\
 &= \text{Var}(h(x)) + \text{bias}(h(x))^2 + \sigma^2 \\
 \text{Expected Error} &= \text{Variance} + \text{bias}^2 + \text{Noise}^2
 \end{aligned}$$

True function is  $y = f(x) + \epsilon$ , where  $\epsilon \sum_i [y_i - h(x_i)]^2$  is normally distributed with zero and standard deviation  $\sigma$ .

Given a set of training examples,  $\{(x_i, y_i)\}$ , we fit an hypothesis  $h(x) = w \cdot x + b$  to the data to minimize the squared error  $\sum_i [y_i - h(x_i)]^2$ .

Now, given a new data point  $x^*$ , we would like to understand the expected prediction error.  $E[(y^* - h(x^*))^2]$ .

For a particular training sample taken from population  $P$ . Compute  $[(y^* - h(x^*))^2]$ . And decompose it into bias, variance and noise.

We have a lemma,  $E[(Z - Z^*)^2] = E[(Z)^2] - Z^{*2}$ .

Variance describes how much  $h(x^*)$  varies from one training set to another.

Bias describes the average error of  $h(x^*)$ .

Noise describes how much  $y^*$  varies from  $f(x^*)$

Using this lemma, we perform the derivation.

**Question 10:**

A linear kernel is a degenerate version of a non linear Gaussian kernel, hence the linear kernel is never more accurate than a properly tuned Gaussian kernel. A properly tuned Gaussian kernel can never perform worse than a linear SVM.

Normalization is necessary because if there is a high difference in the range of values of different features, then their contribution also varies. For example consider two features of the range 0-1, 0-1000. Then the feature 2 contributes more than feature 1 because of higher values, even though feature 1 might be more important.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

There are two different aspects to this — (1) memory-intensive while training, and (2) memory-intensive while prediction.

SVMs are memory-intensive while training, if you are working in the dual space. This is because you need to store an  $N \times N$  kernel matrix, which may be very large if N is large. If you do not store the kernel matrix, then you have to recompute the kernel values repeatedly, making the training much slower.

At prediction time, SVM takes a linear combination of all support vectors. So, if there are a lot of support vectors, you need to store all of them.

**Question 11:**

If the algorithm is suffering from high variance increasing the training data will help and increase the learning rate.

If it is suffering from High bias then increasing training data won't help, as it will lead to under-fitting. So, try getting additional features and decrease learning rate.

Bagging means that you take bootstrap samples (with replacement) of your data set and each sample trains a (potentially) weak learner. It is a way to decrease the variance of your prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same cardinality/size as your original data. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to expected outcome.

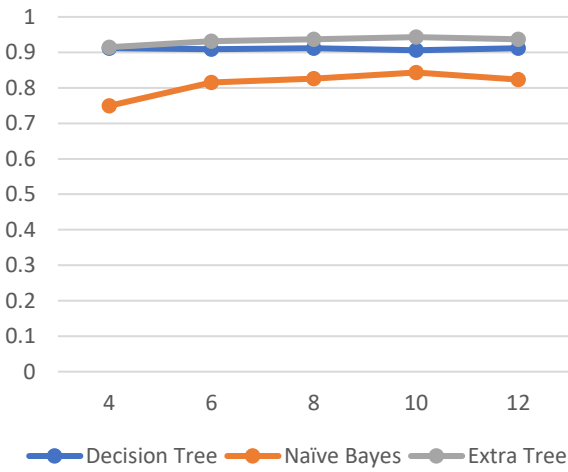
It aims to decrease variance, not bias and is suitable for high variance low bias models (complex models).

Boosting uses all data to train each learner, but instances that were misclassified by the previous learners are given more weight so that subsequent learners give more focus to them during training. It uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.

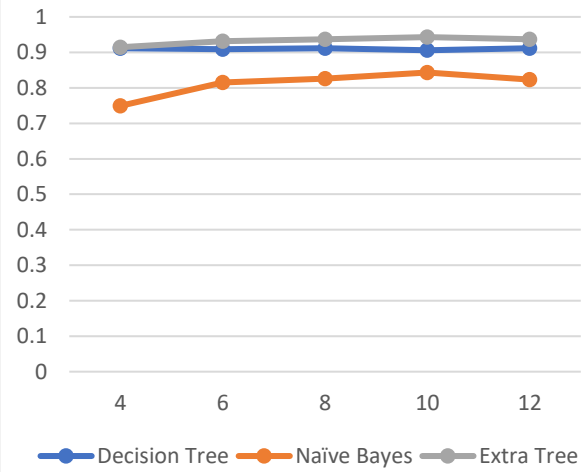
It aims to decrease bias, not variance and is suitable for low variance high bias models.

As the value of k is increased the accuracy increases, but remains constant after  $k = 6$ .

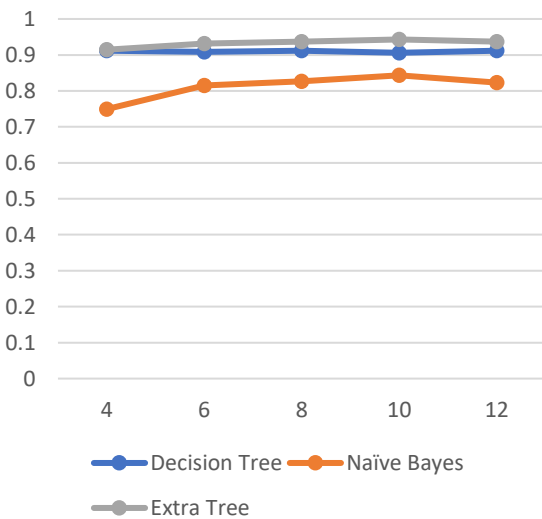
AdaBoost- Ionosphere Data Set



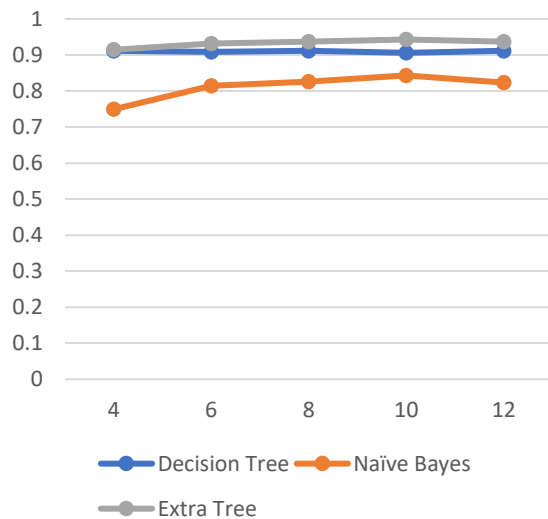
Bagging - Ionosphere Data Set



AdaBoost - Housing DataSet



Bagging - Housing Data Set



For K-Fold 12

Ionosphere DataSet

AdaBoost Classifier

- i) Decision Tree : 0.883190883191
- ii) Naïve Bayes : 0.626780626781
- iii) Extra Tree : 0.85754985755

Bagging Classifier

- i) Decision Tree : 0.911680911681
- ii) Naïve Bayes : 0.823361823362
- iii) Extra Tree : 0.94301994302

Boston Housing DataSet

AdaBoost Classifier

- i) Decision Tree : 0.860398860399
- ii) Naïve Bayes : 0.626780626781
- iii) Extra Tree : 0.85754985755

#### Bagging Classifier

- i) Decision Tree : 0.911680911681
- ii) Naïve Bayes : 0.823361823362
- iii) Extra Tree : 0.937321937322

#### For K-Fold 4

##### Ionosphere DataSet

##### AdaBoost Classifier

- i) Decision Tree : 0.880341880342
- ii) Naïve Bayes : 0.660968660969
- iii) Extra Tree : 0.834757834758

#### Bagging Classifier

- i) Decision Tree : 0.900284900285
- ii) Naïve Bayes : 0.749287749288
- iii) Extra Tree : 0.920227920228

##### Boston Housing DataSet

##### AdaBoost Classifier

- i) Decision Tree : 0.863247863248
- ii) Naïve Bayes : 0.660968660969
- iii) Extra Tree : 0.871794871795

#### Bagging Classifier

- i) Decision Tree : 0.911680911681
- ii) Naïve Bayes : 0.749287749288
- iii) Extra Tree : 0.91452991453