Griffin Klett

Gk2591

COMS 4771 SP21 HW4

4/11/21

1. A question on active learning

a.

The goal of the learning algorithm is to find the best α value from the samples drawn iid from $\mathcal{D}$. Then we can use properties of the VC theorem to prove properties about our f_α classifier.

To get the experimental α / classifier, we need to find the two boundary points where the ouput switches from 0 to 1. Then we can take an average between these two points to get our α

```
---
learn_classifier(S):
    min_x_1 = 1
    max_x_0 = 0

    for s_i in S:
        #if y = 1, check if x value is lower than our known min_x_1
        if s_i.y == 1:
            min_x_1 = min(min_x_1, s_i.x)
        #if y = 0, check if x value is higher than our known max_x_1
        else:
            max_x_0 = max(max_x_1, s_i.x)

    #now take the average of the two values (this would be the range that we are
unsure whether to output 1 or 0, so we take the average)

    alpha = (min_x_1 + max_x_0) / 2

    def classifier (x_input):
        return x_input > alpha ? 1 : 0

    return classifier

----
```

Now we need to prove the following properties:

$O(m)$ runtime:

Our algorithm runs a single loop over S of size n, with no nested loops, so our runtime is linear on the size of data aka $O(m)$.

For any $\epsilon, \delta > 0$ if $m \geq O(\frac{1}{\epsilon^2}\ln\left(\frac{1}{\delta}\right))$ then with probability at least $1 - \delta$ over the draw of S,$err(f_\alpha) \leq \epsilon$

The VC theorem could give us these properties. It states that if:

$$m \geq \frac{C * VC(\mathcal{F}) * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2}$$

Then w/ probability at least $1 - \delta$, then $err(f_m^{ERM}) - err(f^*) \leq \epsilon$

We need to show that

$$\frac{C * VC(\mathcal{F}) * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2} \leq C_1 * (\frac{\ln\left(\frac{1}{\delta}\right)}{\epsilon^2})$$

For some constant C_1. In order to do this, we must show that $VC(\mathcal{F})$ is finite. Start with

d=1

For a single point, x_1, there are only two cases.

If the label $y_1 = 1$, then we can pick any α in the range $[x_1, 1]$, which would classify correctly $f_\alpha(x_1) = 1[x \geq \alpha]$ = 1 (correct)

If the label $y_1 = 0$, then we can pick any α in the range $[0, x_1]$ which would classify correctly $f_\alpha(x_1) = 1[x \geq \alpha]$ = 0 (correct)

More generally, if $x_1 < x_2$ && $y_1 = 1$ && $y_2 = 0$ then we cannot shatter.

d=2

$$x_1 = 0.2, x_2 = 0.4, y_1 = 1, y_2 = 0$$

There is no such f_α which correctly classifies these two points.

(if $\alpha \geq 0.2, x_2\ is\ labeled\ incorrectly; if\ \alpha \leq 0.2, x_1\ is\ labeled\ incorrectly$)

So we can say that $VC(\mathcal{F}) \geq 1, and\ VC(\mathcal{F}) \leq 2$ which bounds $VC(\mathcal{F}) = 1$. Return to VC Thereom.

$$m \geq \frac{C * VC(\mathcal{F}) * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2} \geq \frac{C * 1 * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2}$$

Since C is a constant, it is of the same order and we can say that this satisfies the property of the question:

$$m \geq O(\frac{\ln\left(\frac{1}{\delta}\right)}{\epsilon^2})$$

In our case $f_m^{ERM} = f_\alpha$, and $f^* = 0$ since it is given that $\exists f_\alpha \epsilon\, \mathcal{F}: err(f_\alpha) = 0$

So we can rewrite from VC theorem as $err(f_m^{ERM}) - err(f^*) \leq \epsilon$ to

$$f_\alpha \leq \epsilon$$

Which is the property the question asks for.

The property of the probability $1 - \delta$ is taken directly from VC, without any alteration/substitution needed.

b.

```
learn_classifier(epsilon):
    min_x_1 = 1
    max_x_0 = 0

    while(min_x_1 – max_x_0 > epsilon)
        #get label at midpoint of unknown range
        avg_x = (min_x_1 – max_x_0) / 2
        avg_yval = GetLabelForXValue(avg_x)
        #if label is 1, this is new lower bound for x points w/ label 1
        if avg_yval == 1:
            min_x_1 = avg_x
        #if label is 0, this is new upper bound for x points w/ label 0
        else:
            max_x_0 = avg_x

    #now take the average of the two values (this would be the range that we are unsure whether to output 1 or 0, so we take the average)

    alpha = (min_x_1 + max_x_0) / 2

    def classifier (x_input):
        return x_input > alpha ? 1 : 0

    return classifier
```

First, let us address correctness of the return $err(f) \leq \epsilon$.

The outputted classifier from the algorithm predicts label given the outputted α, which is an average of min_x_1 and max_x_2. This range (min_x_1 – max_x_2) *is the range which we are not sure about the correct label, and the range for which f(x) could predict incorrectly.* So to ensure $err(f) \leq \epsilon$, we need to shrink the range of (min_x_1 – max_x_2) to $\leq \epsilon$. Since our outputted classifier has this quality (otherwise the while loop would not terminate), we can guarantee the correctness $err(f) \leq \epsilon$ with probability 1.

Second, the algorithm makes $O\left(\ln\left(\frac{1}{\epsilon}\right)\right)$ queries.

By using binary search, the window/range (min_x_1 – max_x_2) is halved each time.

Suppose k is the number of iterations.

When k=0, $\epsilon = 1$ since the range of values is [0,1].

$$\frac{1}{2^k} \leq \epsilon$$

$$2^k \leq \frac{1}{\epsilon}$$

$$\ln(2^k) \leq \frac{1}{\epsilon}$$

$$k * \ln(2) \leq \ln\left(\frac{1}{\epsilon}\right)$$

$$k \leq \ln\left(\frac{1}{\epsilon}\right) * (\frac{1}{\ln\,(2)})$$

since $\left(\frac{1}{\ln(2)}\right) \geq 1, we\ can\ rewrite\ as$

$$k \leq \ln\left(\frac{1}{\epsilon}\right)$$

Since each iteration asks 1 query, the above statement also proves that the algorithm makes at most $O\left(\ln\left(\frac{1}{\epsilon}\right)\right)$ queries.

And since we are guaranteed that each query returns an answer in $O(1)$ time, our algorithm runs $\ln\,(\frac{1}{\epsilon})$ times and each loop has O(1) time so the total runtime is $O\left(\ln\left(\frac{1}{\epsilon}\right)\right)$.

c.

```
learn_classifier(S):
    sorted_S = Sort(S)

    left_index = 0
    right_index = len(Sorted_S) -1 #assume zero indexing

    while (left_index < right_index):
        mid_index = (int)(left_index + (right-left)/2)
        x_i = sorted_S[mid_index]
        y_i = GetLabelFromOracle(x_i)
        if y_i == 0:
            left_index = mid_index + 1
        else:
            right_index = mid_index – 1



    #now take the average of the two values (this would be the range that we are
unsure whether to output 1 or 0, so we take the average)
    If (left = 0)
        alpha = sorted_S[left]/2
    else:
        alpha = (sorted_S[left] + sorted_S[left – 1]) / 2

    def classifier (x_input):
        return x_input > alpha ? 1 : 0

    return classifier
```

The key difference in this algorithm is that we start by sorting the samples in S by their x value. We can assume merge sort is used which has a general runtime of $O(n \log n)$. With m data points for our scenario, the runtime for sorting is $O(m \log m)$. Note that this is only done once as part of pre-processing.

After the data is sorted, we run a binary search on the sorted data. By the exact same reasoning as in part b, this has a runtime of $O(\log m)$. Each loop makes one call to the oracle which takes $O(1)$.

So the total runtime is $O(m \log m + \log m)$, which simplifies to $O(m \log m)$

Now for number queries/calls to O:

By using binary search, the window/range (right – left) is halved each time.

Suppose k is the number of iterations, and z is the remaining dataset size. At worst, the algorithm will terminate when z = 1.

$$\frac{1}{2^k} * m = z$$
$$\frac{1}{2^k} * m = 1$$
$$\frac{1}{2^k} \leq \frac{1}{m}$$
$$2^k \leq m$$

$$\log_2\left(2^k\right) \leq \log_2(m)$$

$$k \leq \log_2(m) \sim O(\log(m))$$

generalizing from base two, we can see that even in worst case, our algorithm makes k iterations, and since each iteration we call the oracle once, so we make $O(\log(m))$ calls to O.

For the condition if $m \geq O\left(\frac{1}{\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right)$ then with probability at least $1-\delta$ over the draw of S the algorithm returns f_α s.t. $err(f_\alpha) \leq \epsilon$:

We apply the VC theorem here. Since our classifier model class is the same as in part b, then we know that the VC dimension is the same here as well. So:
$VC(\mathcal{F}) = 1$.

$$m \geq \frac{C * VC(\mathcal{F}) * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2} \geq \frac{C * 1 * \ln\left(\frac{1}{\delta}\right)}{\epsilon^2}$$

Since C is a constant, it is of the same order and we can say that this satisfies the property of the question:

$$m \geq O(\frac{\ln\left(\frac{1}{\delta}\right)}{\epsilon^2})$$

In our case $f_m^{ERM} = f_\alpha$, and $f^* = 0$ since it is given that $\exists f_\alpha \epsilon\, \mathcal{F}: err(f_\alpha) = 0$

So we can rewrite from VC theorem as $err(f_m^{ERM}) - err(f^*) \leq \epsilon$ to

$$f_\alpha \leq \epsilon$$

Which is the property the question asks for.

The property of the probability $1-\delta$ is taken directly from VC, without any alteration/substitution needed.

2. BER / application of large deviation theory

From

$$BER(f) = \frac{1}{2}\left(\mathbb{p}_{x,y\sim\mathcal{D}}[f(x) \neq y|y = 1] + \mathbb{p}_{x,y\sim\mathcal{D}}[f(x) \neq y|y = 0]\right)$$

We break into False positive rate & false negative rate:

$$FPR(f) = \mathbb{p}_{x,y\sim\mathcal{D}}[f(x) \neq y|y = 1]$$

$$FNR(f) = \mathbb{p}_{x,y\sim\mathcal{D}}[f(x) \neq y|y = 0]$$

$$BER(f) = \frac{1}{2}\left(FNR(f) + FPR(f)\right)$$

if both these two estimated parts of BER have ϵ within the range required $\epsilon \in (0,1)$

Then the $BER(f)$ will also be within the required ϵ.

i.e.

$$\overline{FNR}(f) - FNR(f) \leq \epsilon$$

$$\overline{FPR}(f) - FPR(f) \leq \epsilon$$

Yields:

$$BER(f) = \frac{1}{2}\left(FNR(f) + FPR(f)\right) \leq \frac{1}{2}(\epsilon + \epsilon) \leq \epsilon$$

For the above inequality to hold true, we need to estimate our own $\overline{BER(f)} = \frac{1}{2}\left(\overline{FNR}(f) + \overline{FPR}(f)\right)$

And we need our estimate to be within the bounds 99% of the time:

$$\left|\, \overline{BER(f)} - BER(f)\right| = \frac{1}{2}\left(\left|\overline{FNR(f)} - FNR(f)\right| + \left|\overline{FPR(f)} - FPR(f)\right|\right)$$

$$\left|\, \overline{BER(f)} - BER(f)\right| = \frac{1}{2}\left(\left|\overline{FNR(f)} - FNR(f)\right| + \left|\overline{FPR(f)} - FPR(f)\right|\right) \leq \epsilon$$

$$\left|\, \overline{BER(f)} - BER(f)\right| = \frac{1}{2}\left(\left|\overline{FNR(f)} - FNR(f)\right| + \left|\overline{FPR(f)} - FPR(f)\right|\right) \leq \epsilon$$

$$\mathbb{p}\left(\left|\, \overline{BER(f)} - BER(f)\right| \leq \epsilon\right) \geq .99$$

In order to think about how to do this probability guarantee, start by breaking the FPR and FNR out into two events by union bound:

$$\mathbb{P}\left(\left|\overline{FNR(f)} - FNR(f)\right| \geq \epsilon\right) + \mathbb{P}\left(\left|\overline{FPR(f)} - FPR(f)\right| \geq \epsilon\right) \leq .01$$

m is the total # of samples, which is a combination of points where $Y = 1\ \&\&\ Y = 0$

$$m = m_{y=1} + m_{y=0}$$

We also need to know how many samples of each kind are misclassified:

$$m_{y=1,mislabled} = m_{y=1} * \overline{FPR}(f)$$

$$m_{y=0,mislabled} = m_{y=0} * \overline{FNR}(f)$$

This gives us another way of writing $\overline{FNR}(f)$:

$$\overline{FPR}(f) = \frac{m_{y=1,mislabled}}{m_{y=1,}}$$

$$\overline{FNR}(f) = \frac{m_{y=0,mislabled}}{m_{y=0}}$$

Then use the $\mathbb{P}[Y = 0], \mathbb{P}[Y = 1]$,

To determine how large of m_{total}points you need. Note that the final form of the answer is dependent on this because you need to use the probability of $P[Y = 0]$ to guarantee that you will get enough sample points. Use Markov's inequality to show that you have $\geq .99$ of getting enough samples of each type $Y = 1, Y = 0$ to get $FNR(f), FPR(f)$ within the bounds of ϵ

3. Studying k-means

a.

We will find that if we optimize over k and c, we will get undesirable results.

Starting from the typical optimization form:

$$\sum_{i=1}^{n} \min_{j\,\epsilon\{1,\ldots k\}} \left|x_i - c_j\right|^2$$

To:

$$\min_{k\,\epsilon\,\{1,\ldots n\}} \sum_{i=1}^{n} \min_{j\,\epsilon\{1,\ldots k\}} \left|x_i - c_j\right|^2$$

We see that the part of the expression which contributes to the squared distance $\left|x_i - c_j\right|^2$ can take a minimum value of 0 when $x_i = c_i$ for all x_i.

We see next that j ranges from 1 to k (the number of clusters). In our additional optimization over k and c, k can range from $1\ to\ n$ (assuming we impose the max # of clusters as the same # as total datapoints.)

If we set $k = n$, then we will get the minimum squared distance of 0, as each datapoint will be equal to a cluster center.

$$set\ c_i = x_i for\ all\ c$$

$$\sum_{i=1}^{n} \min_{j\,\epsilon\{1,\ldots n\}} \left|x_i - c_j\right|^2 = \sum_{i=1}^{n} \min_{j\,\epsilon\{1,\ldots n\}} |x_i - x_i|^2 = 0$$

So, the minimum value of squared distance is 0, when:

$$k = n$$

$$c_i = x_i\ for\ all\ i\ from\ 1\ to\ n$$

Which is essentially useless information, since it does not group the data in any meaningful way (or does not add any information).

b.

We can see that there are suboptimal (local minima) solutions which Lloyd's method will stabilize at and return that are not the best, global minimum.

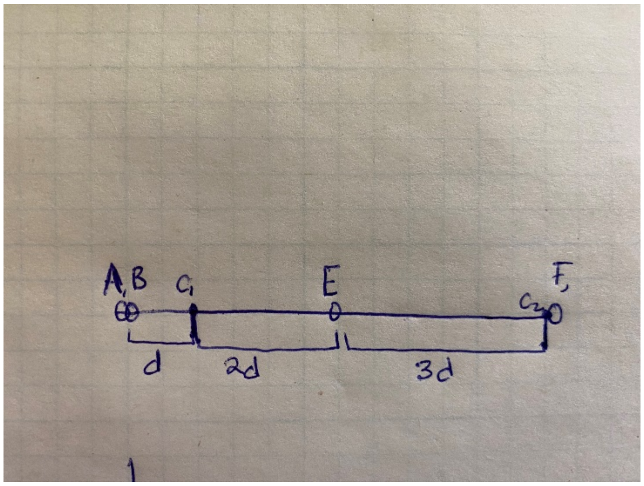
Consider four points $\{A, B, E, F\}\ in\ \mathbb{R}^1$

$A, B = 0,\ \ E = 3d, F = 6d$ where d is some arbitrary positive number.

Suppose that for $k = 2$, the random initialization produced cluster means of

$$c_1 = d, c_2 = 6d$$

Then our setup would appear as follows:



Which would lead to a minimum squared distance of:

$$A\ (c_1 cluster) {:}=\ \ |A - c_1|^2 = |0 - d|^2 = d^2$$

$$B\ (c_1 cluster) \coloneqq\ |B - c_1|^2 = |0 - d|^2 = d^2$$

$$E(c_1 cluster) \coloneqq\ |E - c_1|^2 = |3d - d|^2 = 4d^2$$

$$F(c_2 cluster) \coloneqq\ |F - c_2|^2 = |6d - 6d|^2 = 0$$

$$Total = 4d^2 + d^2 + d^2 = 6d^2$$

After you calculate the new means for the $c_1 and\ c_2$:

$$c_1 = \frac{(A + B + E)}{3} = \frac{0 + 0 + 3d}{3} = d$$

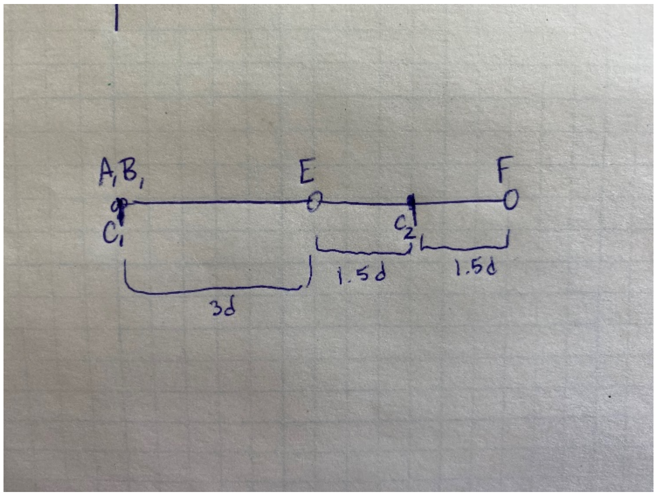
$$c_2 = \frac{F}{1} = \frac{6d}{1} = 6d$$

This would not lead to any further change in the cluster organization or the means.

Now consider an alternative setup where:

$$c_1 = 0$$

$$c_2 = 4.5d$$



Which would lead to a minimum squared distance of:

$$A\ (c_1 cluster) := \ |A - c_1|^2 = |0 - 0|^2 = 0$$

$$B\ (c_1 cluster) \coloneqq \ |B - c_1|^2 = |0 - 0|^2 = 0$$

$$E(c_3 cluster) \coloneqq \ |E - c_1|^2 = |3d - 4.5d|^2 = 2.25d^2$$

$$F(c_2 cluster) \coloneqq \ |F - c_2|^2 = |6d - 4.5d|^2 = 2.25d^2$$

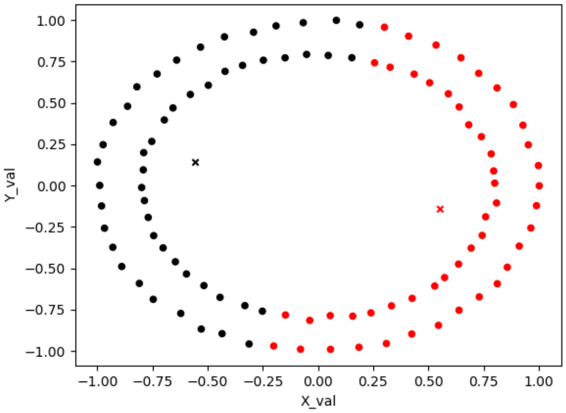
$$Total = 0 + 0 + 2.25d^2 + 2.25d^2 = 4.5d^2$$

This alternate arrangement has a lower total squared distance making it a more optimal solution, and shows that Lloyd’s method does not always return the best and is suboptimal.

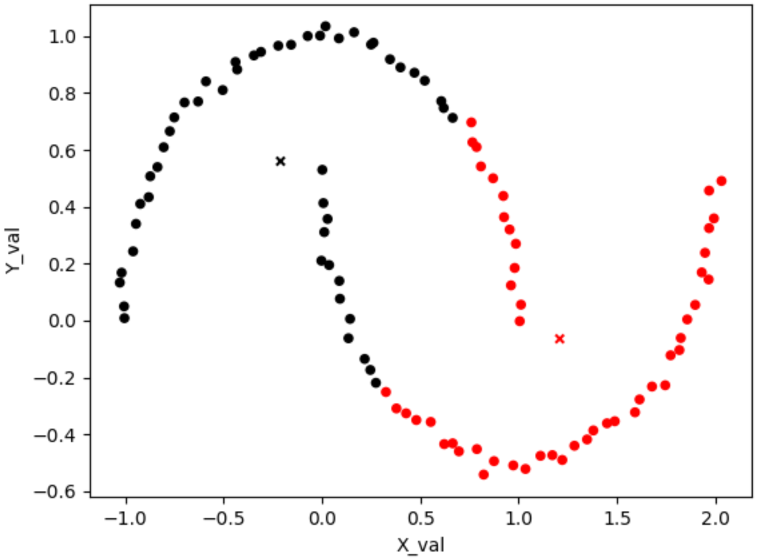
c.

c.i.

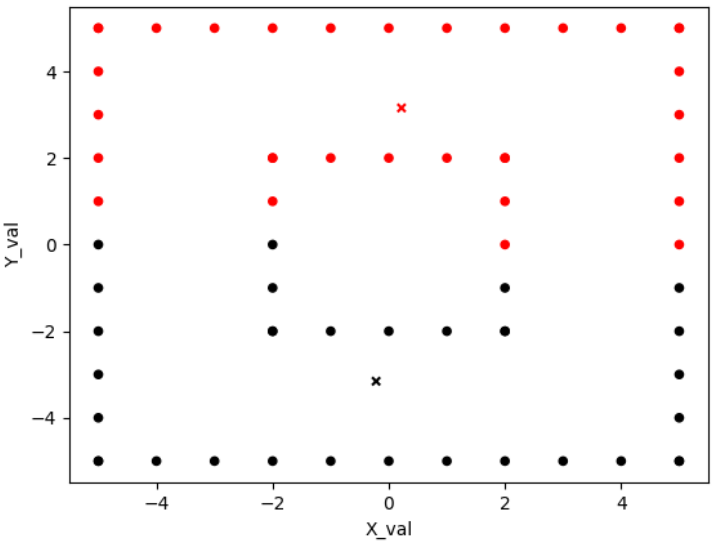
[Code submitted separately]



Another dataset where the kmeans fails is on interleaving curves (sklearn.make_moons):



And finally on rectangle data:


Y_val
X_val
4
2
0
−2
−4
−4
−2
0
2
4

c.ii

Starting with equation 1 and $k = 2$:

$$\sum_{i=1}^{n} \min_{j\ \epsilon\{1,\dots n\}} \left|x_i - c_j\right|^2$$

We can think about an arbitrary point M on the boundary. We know that this point is equidistant from both $C_1\ and\ C_2$, since if it were not, then it would not be on the boundary.

Since it is 2d, we can write:

$m = [\begin{matrix} m_1 \\ m_2 \end{matrix}]$, $c_1 = \begin{bmatrix} c_{11} \\ c_{21} \end{bmatrix}, and\ c_2 = [\begin{matrix} c_{12} \\ c_{22} \end{matrix}]$

So we can say:

$$|m - c_1| = |m - c_2|$$

Square both sides to get in form of L2-norm:

$$|m - c_1|^2 = |m - c_2|^2$$

Expand:

$$(m - c_1)^T(m - c_1) = (m - c_2)^T(m - c_2)$$

$$\left||m|\right|^2 - 2m^T c_1 + \left||c_1|\right|^2 = \left||m|\right|^2 - 2m^T c_2 + \left||c_2|\right|^2$$

$$-2m \cdot c_1\ + \left||c_1|\right|^2 = -2m^T c_2 + \left||c_2|\right|^2$$

$$-2m \cdot c_1 + 2m \cdot c_2 + \left||c_1|\right|^2 - \left||c_2|\right|^2 = 0$$

$$m \cdot (-2c_1 + 2c_2) + \left||c_1|\right|^2 - \left||c_2|\right|^2 = 0$$

Where:

$$w = -2c_1 + 2c_2$$

$$b = \left||c_1|\right|^2 + \left||c_2|\right|^2$$

$$0 = m \cdot w + b$$

This gives us a linear equation for points along the boundary – so the cluster boundary can be defined by this linear equation and is necessarily linear.

c.iii

Show that for any vector $f \epsilon \mathbb{R}^n$,

$$f^T L f = \frac{1}{2}\sum_{ij} w_{ij}(f_i - f_j)^2$$

$$L = D - W = \begin{bmatrix} d_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & d_{nn} \end{bmatrix}$$

$$f^T L f = f^T D f - f^T W f$$

$$f^T D f - f^T W f = \sum_i f_i^2 d_{ii} - \sum_{i,j} f_i f_j w_{ij}$$

Where the diagonal $d_{ii} = sum\ of\ w\ values\ in\ the\ ith\ row$

$$f^T L f = \sum_i f_i^2 \sum_j w_{ij} - \sum_{i,j} f_i f_j w_{ij}$$

Pull out sum over j from first term:

$$f^T L f = \sum_{i,j} f_i^2 w_{ij} - \sum_{i,j} f_i f_j w_{ij}$$

Multiple by 2 and divide by ½

$$f^T L f = \frac{1}{2}\left(2\sum_{i,j} f_i^2 w_{ij} - 2\sum_{i,j} f_i f_j w_{ij}\right)$$

Break the diagonal term down: (switch to summing column wise instead of row wise)

$$2\sum_{i,j} f_i^2 w_{ij} = \sum_{i,j} f_i^2 w_{ij} + \sum_{i,j} f_j^2 w_{ji}$$

Which yields:

$$f^T L f = \frac{1}{2}\left(\sum_{i,j} f_i^2 w_{ij} - 2\sum_{i,j} f_i f_j w_{ij} + \sum_{i,j} f_j^2 w_{ij}\right)$$

Since $w_{ij} = w_{ji}$ because the edges are bidirectional, we can factor out w_{ij}

$$f^T L f = \frac{1}{2}\left(\sum_{i,j} w_{ij}(f_i^2 - 2f_i f_j + f_y^2)\right)$$

Now we have in the form of $x^2 - 2xy + y^2$ which we can refactor as $(x - y)^2$

$$f^T L f = \frac{1}{2} \sum_{i,j} w_{ij} \left(f_i - f_j \right)^2$$

c.iv.

Show that L is a positive semi-definite matrix:

Definition of semidefinite: the Matrix's quadratic form is non-negative:

$$Q(x) = \vec{x} \cdot M\vec{x} \geq 0$$

Here we have:

$$f^T Lf = \frac{1}{2}\sum_{i,j} w_{ij}\left(f_i - f_j\right)^2$$

Each value of w_{ij} is ≥ 0 since the contents are an indicator function (1 or 0).

Similarly, the values of $(f_i - f_j)$^2 are guaranteed to be non-negative since it is being squared.

Thus we can think about $f^T Lf = \frac{1}{2}\sum_{i,j}(positive/zero)(positive/zero)$

So

$$f^T Lf \geq 0$$

And so L is positive semi-definite.

Symmetric:

Defn: For symmetric matrix M, $M_{ij} = M_{ji}\ \forall i,j$

Diagonal terms: $L_{ij} = L_{ji}$ (same by definition)

Non-Diagonal terms:

$$L = D - W$$

D terms are zero on non-diagonal. For w:

$$w_{ij} = w_{ji}$$

because the edges are undirectional: (There exists an edge between two points or not). $w_{ij} = 1[\exists edge\ between\ v_i\ and\ v_j in\ G_r]$

so:

$$L_{ij} = D_{ij} - W_{ij} = 0 - W_{ij}$$

$$L_{ij} = -w_{ij} = -w_{ji} = L_{ji}$$

So L is both symmetric and positive semi definite.

$$f^T L = [f_1 \quad \dots \quad f_n]\begin{bmatrix} d_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_{nn} \end{bmatrix} - \begin{bmatrix} 0 & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & 0 \end{bmatrix}$$
$$= [f_1 d_{11} + \cdots + f_n w_{n1} \quad \dots \quad f_1 w_{1n} + \cdots + f_n d_{nn}]$$

$$f^T Lf = [f_1 d_{11} + \cdots + f_n w_{n1} \quad \ldots \quad f_1 w_{1n} + \cdots + f_n d_{nn}]\begin{matrix} f_1 \\ [\ldots] \\ f_n \end{matrix}$$

$$f^T Lf = f_1(f_1 d_{11} + \cdots + f_n w_{n1}) + \cdots + f_n(f_1 w_{1n} + \cdots + f_n d_{nn})$$

c.v

Defn of eigenvector:

$$AX = \lambda X$$

In our case:

$$L\mathbb{I}C_k = \lambda \mathbb{I}C_k$$

$$L\mathbb{I}C_k = (D - W)\mathbb{I}C_k = D\mathbb{I}C_k - W\mathbb{I}C_k$$

It's easiest to look at one specific row (which corresponds to a point v_i)

Two cases:

First if $v_i \in C_{k_j}$:

$$(D\mathbb{I}C_k)_i - (W\mathbb{I}C_k)_i = d_{ii} - \sum_j w_{ij} = \sum_j w_{ij} - \sum_j w_{ij} = 0$$

Since $d_{ii} = \sum_j w_{ij}$

So:

$$(L\mathbb{I}C_k)_i = 0 = (\lambda C_k)_i \; where \; \lambda = 0$$

The second case is:

If $v_i \notin C_{k_j}$:

$$(D\mathbb{I}C_k)_i - (W\mathbb{I}C_k)_i = 0 - 0$$

So:

$$(L\mathbb{I}C_k)_i = 0 = (\lambda C_k)_i \; where \; \lambda = 0$$

Since $C_{k_i} = 0$

So in all cases ($\forall i \; in \; 1 \leq i \leq k$):

$$L\mathbb{I}C_k = \lambda \mathbb{I}C_k$$

$$where \; \lambda = 0$$

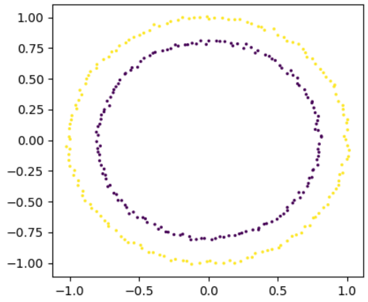
So the vectors $C_1 \; to \; C_k$ are (unnormalized) eigenvectors.

c.vi

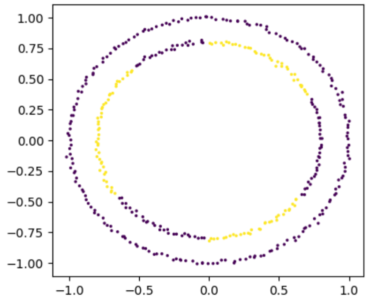
Circle dataset:

The output depends on what number of nearest neighbor to use (as well as size of n). If the number of neighbors is too low or high, the cluster assignment is very poor. For my dataset (with n=300, and number of neighbors = 4), the clustering is greatly improved.

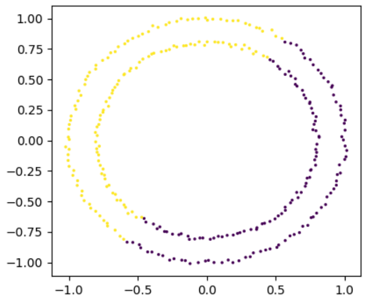
G_r = 4



G_r = 2

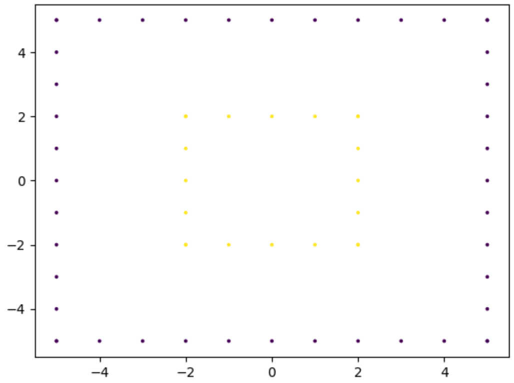


G_r = 50

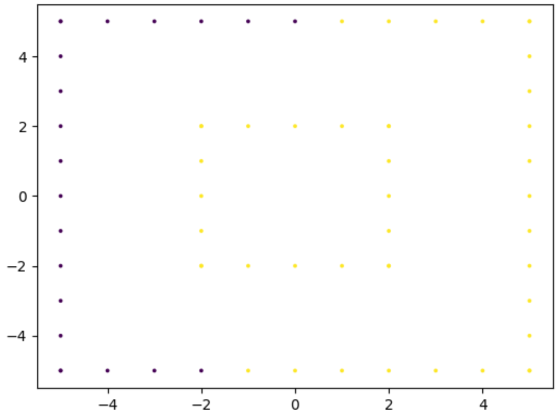


And finally with rectangles:

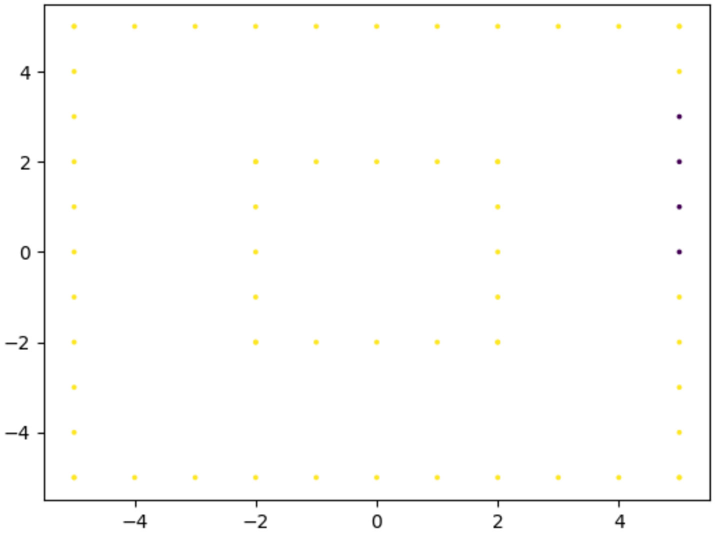
G_r = 3 (nsamples =64)



G_r = 6 (nsamples =64) (fails):



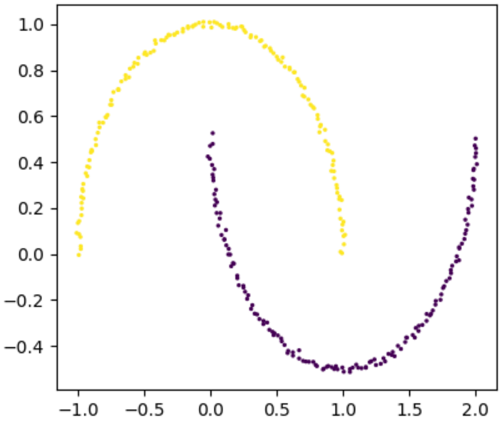
G_r = 1 (nsamples = 64) (fails):



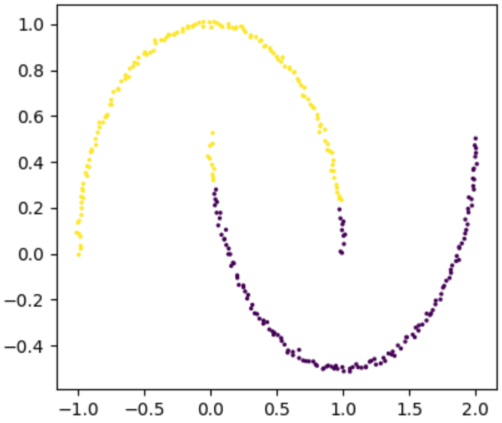
This is expected. If G_r is too large, then even in the transformed representation, it is best to split the data radially across the circle. Similarly, if G_r is too small, then the transformed space splits are not representative enough of what points are actually near others (since the Gr and W matrices are too sparse). If G_r is too small, then there are too many connected components – whereas if G_r is too big, then there are too few connected components.

For the interleaving parabolas, it is a similar situation:

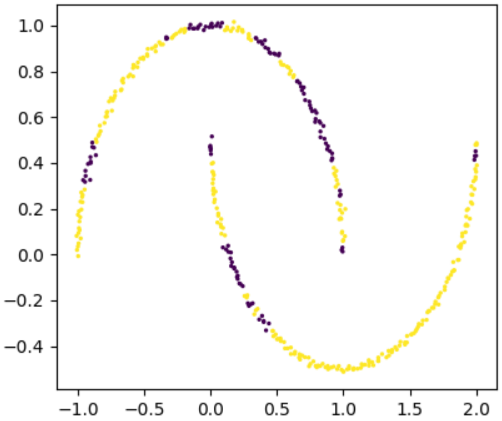
G_r = 4:



G_r =50:



G_r = 2:


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0