# Locality Sensitive Hashing(LSH) for Similar DNA strings

Tim Hearn

# Relevant Code:

- Github Repo:
  - https://github.com/newalchemy/locality_sensitive_hashing
- Google Colab Notebook:
  - https://github.com/newalchemy/locality_sensitive_hashing/blob/main/FinalLSHCodeBasicAndGuidedRandomSampling.ipynb
- Google Colab notebook is a superset of the Github repo code.
  - Github code contains only the data generation process, LSH table, and some offline scripts for analysis.
  - Google Colab code actually uses these tools using parallelization code.

# Problem Statement

- Given a corpus of M DNA sequences each of length N and a query string q of length N, find all DNA sequences within edit distance R from q.
  - A DNA sequence is defined as a string of length N composed of characters picked from the following alphabet:
    - {‘A’, ‘T’, ‘G’, ‘C’}
  - Edit distance is defined as the Levenshtein distance between the two strings. It is a count of the minimum number of insertions, deletions, and substitutions necessary to turn the first string into the second.
- Use Locality Sensitive Hashing to solve this problem

# Problem Statement - Parameter List

M = # of DNA sequences in corpus

N = Length of DNA sequences

R = Maximum desired edit distance between any given query string and its return.

S = Length of longest shared substring / subshingle (used interchangeably) between any two given strings, s1 and s2

# Locality Sensitive Hashing

- Given a set of data, build a hash table such that points which are close to each other are more likely to be hashed to the same bucket to enable fast lookup time based on distance.
- “Close” is defined by the distance metric of the data.

# DNA Strings - Shingling

- *k-Shingling* a DNA string is defined as producing a sequence all substrings of a prespecified length k within the string.
- Example:
  - For the sequence ‘ACTGCAAA’ the 4-shingling set is {‘ACTG’, ‘CTGC’, ‘TGCA’, ‘GCAA’, ‘CAAA’}

# DNA Sequence Edit Distance - Key Theorem

- **(1) Given a set of strings each of size N if 2 strings have an edit distance no greater than R then that means they have at least one shingle of matching characters of size at least S=floor(N/(R+1)) with probability 100%.**
- This theorem will be referred to as (1) from this point on
- This will be proven on the next slide
- This theorem only produces a relevant LSH algorithm for cases where N >> R.
    - This is due to properties of the curse of dimensionality, to be discussed later

# DNA Edit Distance - Key Theorem Informal Proof

- Proof by describing the strings necessary, given a particular edit distance, to produce the shortest possible longest k-shingle
- Observation: **A sequence must have an equal number of insertions and deletions to produce a sequence of length N.**
  - Simple Example: Suppose two sequence, N=600 and R = 2. To produce the shortest possible longest common k-shingle between the sequence, one needs to place a required substitution at indices 200 and 400. This will produce a longest common k-shingle of 200.
- In general, to produce the shortest longest possible common k-shingle, one needs to put a substitution at indices (i*(1/(R+1)) for i between 1 and R inclusive.
- One can replace substitutions with insertions and deletions as desired, so long as every insertion has exactly one deletion.

# Locality Sensitive Hashing Scheme

- Notice that each DNA sequence can be S-Shingled.
- For space constraints on implementation, each sequence and shingle can be given a unique id.  This presentation uses this as an implied fact.
- Observe that a mapping can be produced between a sequence and its list of Shingles of {Sequence -> [Shingles]} for each sequence.  For a corpus of DNA sequences, this mapping can be reversed such that:
- {Shingle -> [Sequences]}
- This mapping is the locality sensitive hashing scheme.
    - ***Note that this LSH scheme has guaranteed perfect recall based on theorem (1)***
        - Precision is probabilistic and based entirely on the corpus of DNA strings.

# Locality Sensitive Hashing Scheme - Time Analysis

- Insertion/Deletion - Shingle each sequence, and update the {Shingle -> [Sequence]} mapping.  Notice there are N-S shingles of length S, so this operation takes O(N-S) = O(N-N(1/(R+1)) = O(N) time.
- Query:  Shingle each sequence, look up each shingle in the {Shingle -> [Sequence]} mapping.  Fetch the list of sequences associated with each shingle.  Runtime analysis is the same as above:  O(N)
    - These runtime analysis are based on the assumption of a perfect hash function which can hash a string to an ID in O(1) time, and a perfect hash table implementation which can query an ID in O(1) time.

# Locality Sensitive Hashing Scheme - Space Analysis 1

- Any implementation can store a mapping between {ID -> Sequence} and {ID -> Shingle}. The size of the {ID -> Sequence} mapping is $O(MN)$ space complexity
- For the {ID -> Shingle} database, recall that each document has N-S shingles.
- Assuming no overlapping shingles with M Sequences, the {ID->Shingle} Database will be $O(M(N-S))$ space complexity.

# Locality Sensitive Hashing Scheme - Space Analysis 2

- The {Sequence -> [Shingles]} and {Shingle -> [Sequences]} follow a similar logic.
- {Sequence -> [Shingles]}
  - Exactly M keys, and each sequence has N-S shingles. Therefore, space complexity is $O(M(N-S))$
- {Shingle -> [Sequences]} aka. [The LSH table]
  - There are O(M(N-S)) shingles, and M documents. **Each sequence will appear O(N-S) times.**
  - Therefore, space complexity is $O(M(N-S)) + O(M(N-S)) = O(M(N-S))$
- Notice that since M >> N in nearly all use cases, $O(M(N-S)) > O(N(N-S)) > O(N^2+N*S^2)$
  - This makes space complexity greater than $O(N^2)$

# Locality Sensitive Hashing Scheme - Space Analysis Commentary - Bitwise level implementation

- Notice that in most modern computing languages, a character is 1 byte (8 bits). **This is entirely unnecessary for this problem.**
- Use the following mapping to reduce each character to 2 bits:
  - A -> ‘00’, C -> ‘01’, G -> ‘10’, T -> ‘11’
  - Each byte will contain exactly 4 characters except for the last one.
  - Will need an additional byte which contains a count of how many characters the last byte has.
- Each DNA string and shingle stored will now be (ceiling(N/4) + 1) bytes using this method, instead of N.
  - Only relevant for very large datasets due to the fact that you can create a unique integer/float ID for each shingle **and** each sample
- Space complexity of $O(M(N\text{-}S))$ is the only known undesirable feature of this algorithm.

# Experimental Proof of correctness - Python prototype

- Fully implemented LSH scheme prototype in python to prove correctness of approach using synthetic data.
  - Implemented simple class using python dictionaries, created utils module for functions to be outside this class, and used a ‘Two Way Dictionary’ class from stackoverflow to implement a bidirectional dictionary for the maps which contain IDs.
  - Class contains all mappings discussed previously:
    - {ID -> Sequence}, {ID -> Shingle}, {Sequence -> [List of Shingles] }, {Shingle -> [List of Sequences]}
  - 3 Different Sampling techniques to create synthetic data to prove precision and recall
- Data analysis done in parallel using Google Colab.
  - **Notice it takes O(M*N^2) to compute the recall for any given query.**
  - This is because the code needs to compute the O(N^2) edit distance computation M times,
- Plots and Final analysis are computed using offline script located in github repository

# Experimental Proof of correctness - Google Colab

- Implemented on Google Colab to use cloud computing resources for high volume datasets
  - $10 per month subscription buys use of 100 “Compute Units” per month, and an additional $10 gives access to 100 more if you run out within a month.
  - Gives access to high RAM CPU, GPU, and TPU processors.
  - The speed of TPU processors are 15-30 times faster than that of CPU but obviously consume more “Compute Units”
    - Will consume approximately 1-5 Compute units to use a TPU generate an M=50,000 to M=100,000 dataset and compute precision and recall. Nearly all of this cost comes from computing recall using parallelization.
    - Obvious usage pattern: Test, develop, and verify output using smaller M sizes on CPUs, then create larger, more experimentally relevant, ‘runs’ using TPUs.
- https://colab.research.google.com/

# Data Generation vs. Sampling Technique

- Mechanically speaking, the test data used is synthetic data from a data generation process.
- However, ‘sampling technique’ is a far more accurate conceptual description of the implementation than ‘data generation process’ and provides a far more expansive understanding of the requirements and goals of this study.
  - **Definition: The universe of all DNA strings of length exactly N is defined as DNA(N). It has exactly 4^N elements.**
  - Consider the fact that this analysis is attempting to determine properties of the universal precision *p* and recall *r* across all of DNA(N).
  - The data generation process, therefore, is generating a very small sample from this subspace to determine properties of *p* and *r* across subspaces from this universe with given properties.
  - Using the proper sampling techniques, we can discuss properties of *p* and *r* inherent to the algorithm, and how *p* and *r* vary based on inherent subspace propertoies
- ‘Sampling Technique’ will be the terminology used in this presentation, although it is synonymous with ‘Data Generation Process’

# Data Generation vs. Sampling Technique cont.

- Another way of thinking about the goals of this study is to consider that we are attempting to determine estimators of *p* and *r* of this algorithm. Therefore, for any given sample from DNA(N), *sp* and *sr* are estimators with a particular distribution.
  - These estimators will suffer from sampling bias no matter what based on the inherent limitations of the code used to generate this synthetic data.
  - This sampling bias can be discussed as its own topic, **and different sampling techniques can be developed depending entirely on the type of sampling bias you want to study.**
- Therefore, this study is to determine universal properties of *p* and *r* and demonstrate with reasonable certainty that:
  - *r* is 1 across any subsample.
  - The exact conditions where *sp* < *1* can be well-defined.
    - **Key Hypothesis:** ***For a given query DNA string q and parameter R, the algorithm has a far higher probability of returning a sequence s as edit_distance(q,s) -> R for all cases edit_distance(q,s) > R.***
    - Meaning: ***The probability that s will be returned by the algorithm decreases exponentially with as edit_distance(q,s) grows.***

# change_2Z_edit_distance_in_DNA_seq Similar Sample Edit distance Function

- ***Will be referred to as simply 'similar sample edit distance function' throughout this presentation.***
- Python function named change_2Z_edit_distance_in_DNA_seq(...)
- Consider a string s1 and recall the property that any other DNA string s2 of the same length which is not the same as s1 must have an equal number of insertions and deletions computed in its edit distance.
- Created python function which, given a DNA Sequence s1 and an integer Z, makes Z pairs of insertions and deletions iteratively and at random throughout the sequence.
  - Notice that these indices are chosen at random and could 'collide', resulting in an edit distance less than 2N.
  - This function will generate a new string with edit distance from s1 in range of [Z, 2Z] with a probability distribution based on Z, where PR(new edit distance = 2Z) -> 1 as Z -> 1. Based on experimental data:
    - N = 5 -> PR(new edit distance = 2Z) = ~90%
    - N = 10 -> PR(new edit distance = 2Z) = ~70%

# New DNA Sequence - Code Snippet

```
        new_char = int_to_dna_letter(random.randint(0,3));
        inx_to_delete = random.randint(0, seq_length);
        #If the distance between the two indicies is no more than 1 then you're performing substitution, not insert/delete swap
        while (abs(inx_to_delete - inx_char_to_change) <= 1):
            inx_to_delete = random.randint(0, seq_length);

        if (inx_to_delete < inx_char_to_change):
            out_dna_seq = out_dna_seq[:inx_to_delete] + out_dna_seq[inx_to_delete + 1:inx_char_to_change] + new_char + out_dna_seq[inx_char_to_change:seq_length];
        else:
            #deleting the last character is a special case
            if (inx_to_delete == seq_length):
                out_dna_seq = out_dna_seq[:inx_char_to_change] + new_char + out_dna_seq[inx_char_to_change:seq_length - 1];
            else:
                out_dna_seq = out_dna_seq[:inx_char_to_change] + new_char + out_dna_seq[inx_char_to_change:inx_to_delete] +  out_dna_seq[inx_to_delete + 1:seq_length];

return out_dna_seq;
```

- This is the relevant code snippet for the function *change_2Z_edit_distance_in_DNA_seq*
- Code performs above task for *num_changes_to_make* iterations
- Notice indices chosen can very easily collide with previous iterations, **so this makes the actual edit distance between the sequence and its origin returned probabilistic in range between [Z, 2Z].**
    - The distribution of the return is based entirely on Z.  The only fact determined for this study was edit_distance_returned -> 2Z as Z-> 1 from above.

# Sampling Techniques - Summary

- 3 Sampling techniques to prove consistency of results
- ***Sampling Technique 1 - Basic Random Sampling***
  - Choose M samples entirely at random.
  - Affected by curse of dimensionality
  - Good for proving “needle in a haystack” test case (discussed later).
- ***Sampling Technique 2 - Guided Random Sampling***
  - Choose *seeds* <<< M number of seeds each entirely at random
  - Build clusters within edit distance R from each seed.
    - Each cluster will have about floor((M - seeds)/(R + 1)) samples within edit distance R from the original seed.
  - Determine precision and recall for each seed.
    - Does the algorithm find the entire cluster?  Does it find sequences outside of its cluster?
- ***Sampling Technique 3 - Centered Random Sampling***
  - Contrived Sampling Technique to analyze precision performance and prove that precision performance of algorithm degrades gracefully.
    - More specifically, that there is an upper bound based on R for all sequences returned by the algorithm, **including those greater than R.**
  - Choose one seed entirely at random(named ‘center seed’), and a parameter E such that E >> R.
  - Choose sequences within edit distance E from R, **choosing about equal amounts of sequences for each edit distance between 1 and E**
  - Analyze precision and recall performance for center seed **only**.
- More thorough discussion and analysis of each sampling technique to follow

# Sampling Technique 1:  Basic Random Sampling - Experiment Definition

- 5 Experiments
- Parameters for each experiment:
  - M = 100,000
  - N = 1,000
  - R = 10
  - S = 90 = floor(N/(R+1))
- **Experiment Definition**:  1,000 times for each experiment:
  - Pick a random sample s1, and generate a query DNA sequence *q* no more than R edit distance away from it using the similar sample edit distance function discussed previously.
  - Determine the precision and recall of that query.
  - Aggregate the data across experiments to show results

# Sampling Technique 1:  Basic Random Sampling - Results

- 5 Experiments
- Parameters for each experiment:
  - M = 100,000
  - N = 1,000
  - R = 10
  - S = 90 = floor(N/(R+1))
- 1,000 times for each experiment:
  - Pick a random sample s1, and generate a query DNA sequence *q* no more than R edit distance away from it using the similar sample edit distance function discussed previously.
  - Determine the precision and recall of that query.
- **Result:**  Each and every query had precision and recall 1.0, **and exactly 1 sample returned - the original string s1.**
- **Analysis:**  This experiment proves the curse of dimensionality is present in the space DNA(N) described previously.

# Curse of dimensionality - Definition

- “The curse of dimensionality is a phenomenon that occurs when the distance between two randomly selected data points increases dramatically as their dimensionality increases. This can cause the distance between all pairwise points to become equal. “
- Copied from:  google / various sources
    - https://www.google.com/search?q=curse+of+dimensionality+distance+of+points&sca_esv=a1240fbab318ff70&sca_upv=1&ei=_KzEZuXCEqv9ptQP-JjBsQo&ved=0ahUKEwjlsLPE6IOIAxWrvokEHXhMMKYQ4dUDCA8&uact=5&oq=curse+of+dimensionality+distance+of+points&gs_lp=Egxnd3Mtd2l6LXNlcnAiKmN1cnNlIG9mIGRpbWVuc2lvbmFsaXR5IGRpc3RhbmNlIG9mIHBvaW50czIFECEYoAEyBRAhGKABMgUQIRigATIFECEYoAEyBRAhGKsCMgUQIRirAkisElDtA1jBEXABeACQAQCYAZ0BoAGiDqoBBDEzLja4AQPIAQD4AQGYAhOgAosOwgIKEAAYsAMY1gQYR8ICDRAAGIAEGLADGEMYigXCAgoQABiABBhDGIoFwgIFEAAYgATCAgsQABiABBiRAhiKBcICBhAAGBYYHsICCBAAGBYYHhgPwgILEAAYgAQYhgMYigXCAggQABiABBiiBMICBRAhGJ8FmAMAiAYBkAYKkgcEMTIuN6AH8YkB&sclient=gws-wiz-serp

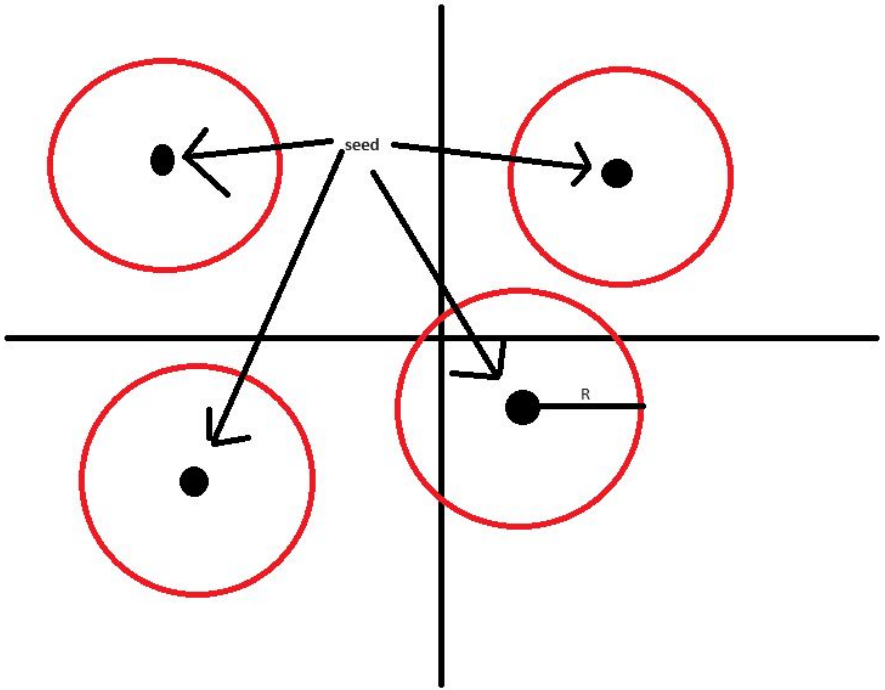
# Curse of dimensionality - Discussion

- The curse of dimensionality applies to DNA, where for s1, s2, in DNA(N), dist(s1,s2) = edit_distance(s1, s2) aka. ‘R’
  - For N=1000, the PR(edit_distance(s1,s2)) converges to the range [500, 540].
    - Separate function on the class which contains all dictionaries for this data structure was implemented to prove this via all pairs comparison of edit distance of all strings. **Note the runtime of $\mathbf{O(M^2N^2)}$.**
  - A relevant corollary is that PR(longest_common_shingle(s1,s2)) converges to the range [8,12].
- This is relevant to this algorithm because in the case where:
  - S = floor(N/(R+1) ⇒ 12 = floor(1000/(R+1)) ⇒ R = 83, the algorithm becomes obsolete due to the curse of dimensionality.
  - ***The algorithm only produces relevant results, therefore, for parameter sets where N >> R .***

# Sampling Technique 2:  Guided Random Sampling - Intuition

- Aka, ‘Clustered’ Random Sampling
- Generate a set of random DNA sequences and generate clusters around those sequences with edit distance R.
- Determine precision and recall for each seed from the set of seeds.
    - Does the algorithm return the entire cluster for each seed?
    - Does the algorithm return DNA sequences from other clusters?

# Sampling Technique 2:  Guided Random Sampling - 2D Visualization




- Each seed is at the center of its own cluster
- Each cluster is of size R

# Sampling Technique 2:  Guided Random Sampling - Precise definition

- Given M, N, R, pick the following new parameter:
  - The number of initial random dna seeds *seeds* parameter.
- Pick *seeds* random DNA sequences to start.
  - Denote *steps* = *math.floor(R/2)*
  - Compute *cluster_size* = *(M - seeds)/seeds*
- For each seed generated at random, pick *cluster_size* new DNA sequences within edit distance R from the seed.
- If there is any remainder (M - seeds*cluster_size + seeds), pick seeds at random and add DNA sequences to their cluster.

# Sampling Technique 2:  Guided Random Sampling - Experiment design

- Across 5 Experiments (4 for the R=50 case):
  - M = 100,000
  - N = 1,000
  - R = {10, 50}
  - S = {90, 19} = {floor(N/(R+1))}
  - Num_dna_cluster_seeds = 500
- **Experiment definition**:  Determine the precision and recall for each of the 500 seeds of each experiment and aggregate across all datasets.

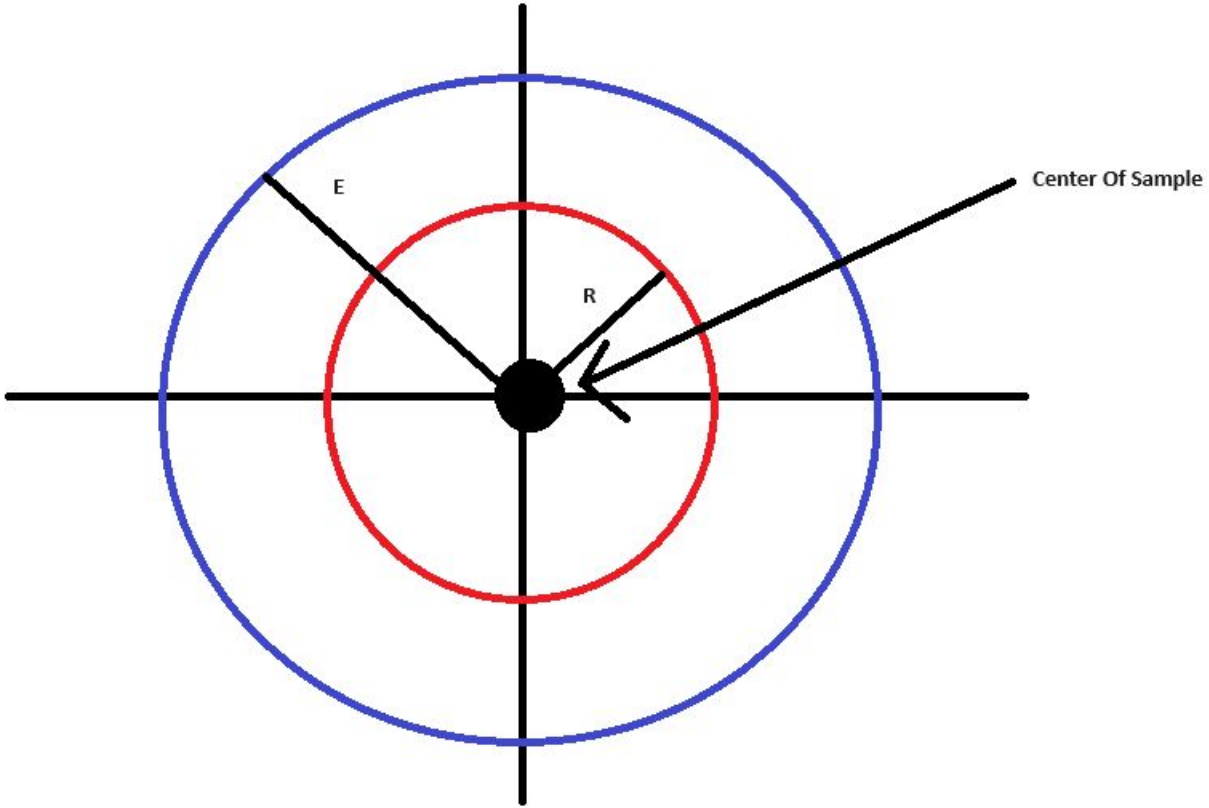
# Sampling Technique 2:  Guided Random Sampling - Results

- Across 5 Experiments:
  - M = 100,000
  - N = 1,000
  - R = {10, 50}
  - S = {90, 19} = {floor(N/(R+1))}
  - Num_dna_cluster_seeds = 500
- **Results:**
  - **R = 10:**  Perfect precision and recall for all samples due to the curse of dimensionality and curse of dimensionality limit(N) >> R
  - **R = 50:  Nearly** perfect precision and perfect recall across all samples.

# Sampling Technique 3:  Centered Random Sampling - Intuition

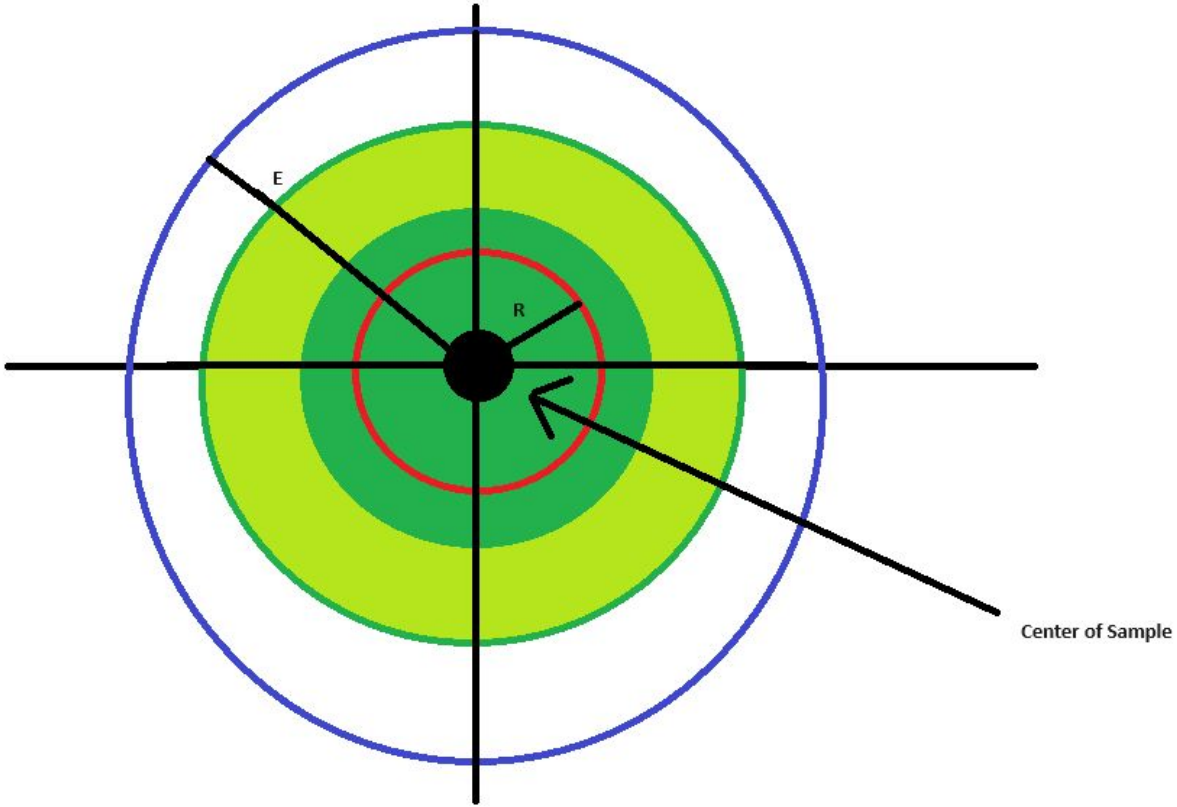
- Pick one DNA sequence at random and a new parameter *max_edit_distance* = *E* where *E* >> R.
- The entire dataset will be a cluster where all sequences are within *E* of the cluster.
- The only sample query used will be the center point of the cluster.
    - This sampling technique is used to prove precision and recall far more thoroughly.
    - Does the algorithm retrieve the entire cluster?  How many points outside the cluster does it retrieve?

# Sampling Technique 3:  Centered Random Sampling - 2D Visualization



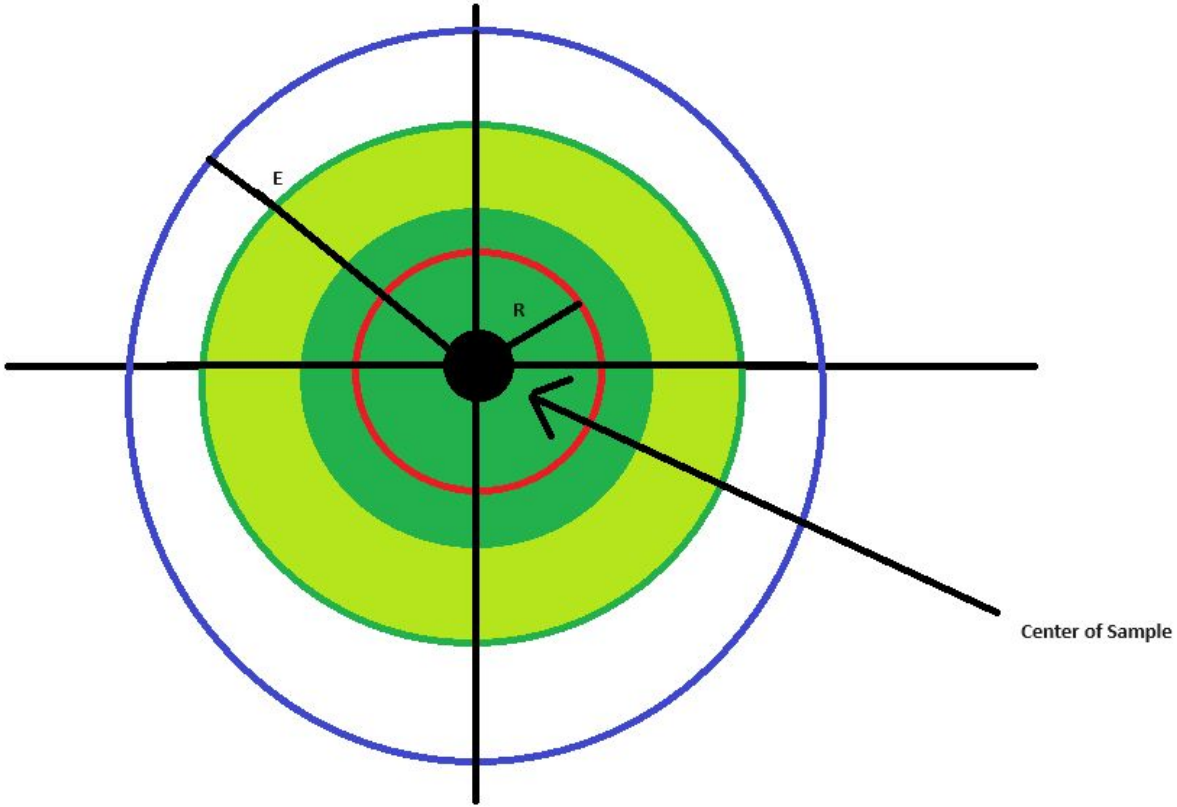

- All points are within distance E from sample center.
- Since E >> R, R represents a subcircle of points within E.

# Sampling Technique 3:  Centered Random Sampling - 2D Visualization




- Conjecture:  The LSH algorithm returns all points within the dark green circle with 100% probability, and within the light green circle with between 0 and 100% probability.
- **Conjecture will be proven with this data sampling technique**
- The size of the light green area grows with R

# Sampling Technique 3:  Centered Random Sampling - 2D Visualization




- **Conjecture:**  The size of the light and green areas each grow with R
- Note that by construction of the algorithm, the dark green area will always supercede the red circle.

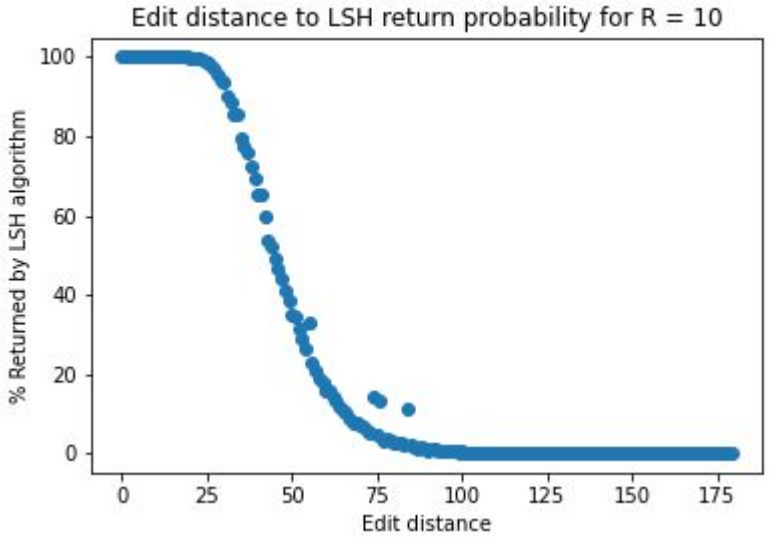
# Sampling Technique 3:  Centered Random Sampling - Precise Definition

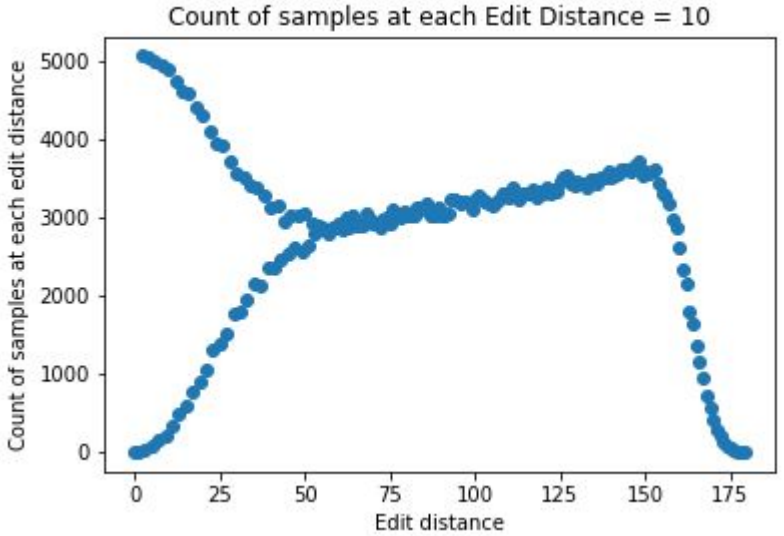
- Generate random seed (this will be *center_dna_sequence*)
- Compute *num_steps = floor(max_edit_distance/2)*
- Compute *size_of_each_step = floor((M - 1)/num_steps))*
- For each step 1… *num_steps* generate *size_of_each_step* random DNA sequences using the similar sample edit distance function.
- For *remainder = M – 1 - (num_steps * size_of_each_step)* strings, generate new strings using a random number of steps between 1 and *size_of_each_step*.

# Sampling Technique 3:  Centered Random Sampling - Experiment Design

- For each R = 10, 20, and 50, set *num_changes_seq* parameter for change_2Z_edit_distance_in_DNA_seq (..) function discussed previously to [100, 100, 150] respectively
- Create a sample size of 50001.
- Perform LSH search only on center of the sample.  Determine precision and recall for center point.
- Find the percentage of points returned at each edit distance for LSH algorithm.
- ***Aggregate these results across 10 different runs ("experiments") for R=10,20, and 50.***

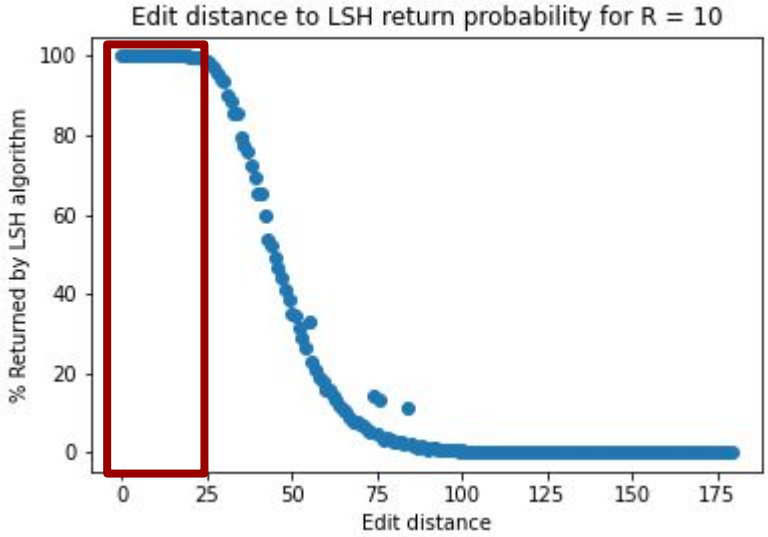
# Sampling Technique 3: Centered Random Sampling - Data R = 10

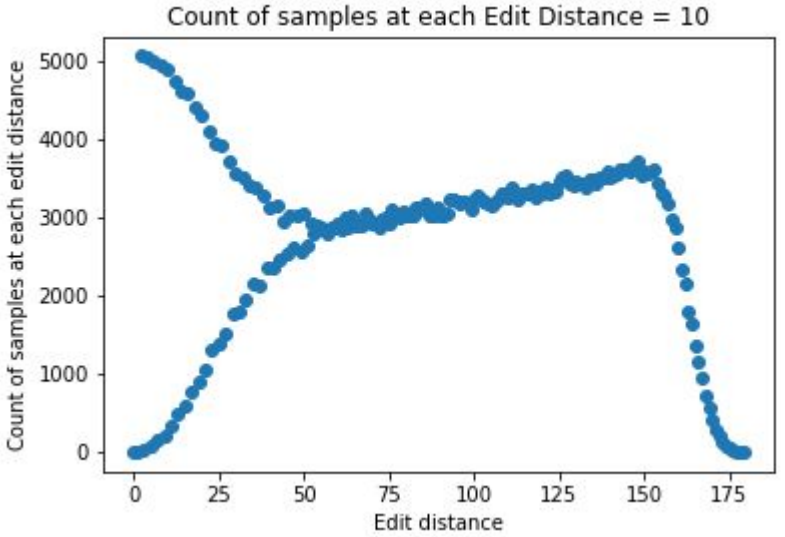



- % LSH returned by Edit distance shows smooth S function (Right plot)
- Count of edit distance from center(Left plot) shows peculiar behavior to be explained on later slide based on the construction of change_2Z_edit_distance_in_DNA_seq
- **Nonetheless, the count of samples at each edit distance shows the experimental design is sufficient for proving the relevance of left hand plot**

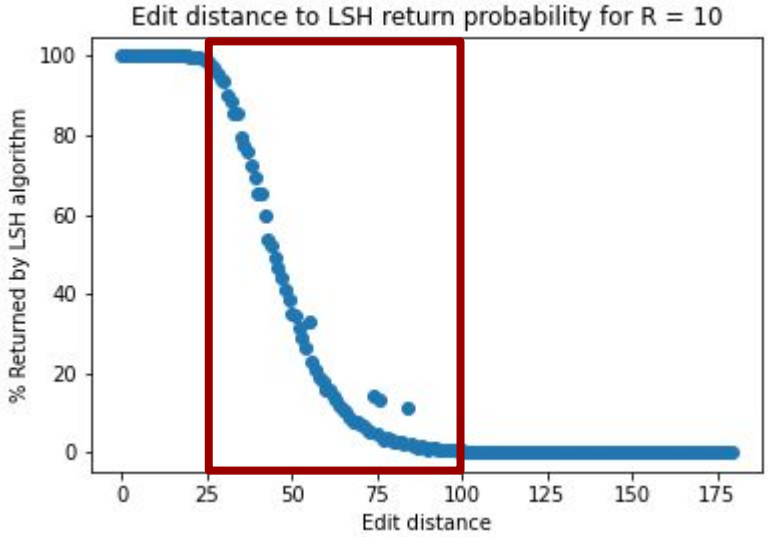
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

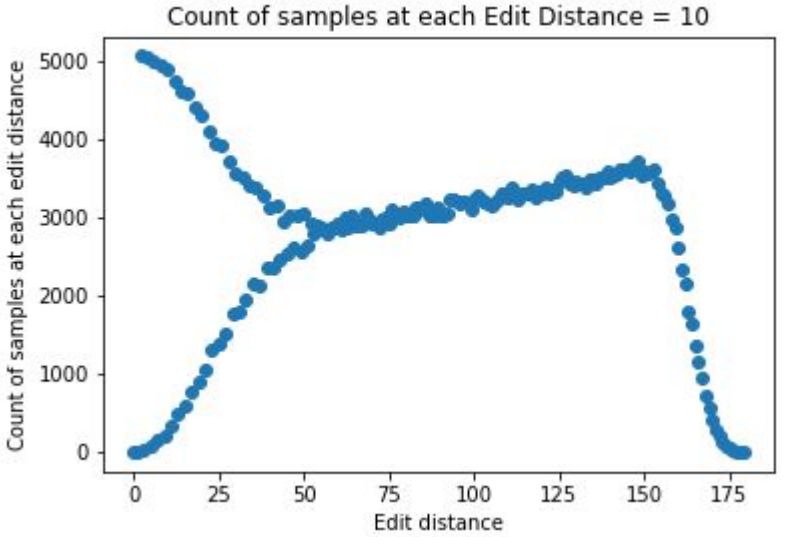



- When R = 10, the algorithm returns 100% or near 100% for all samples of true_edit_distance < 25

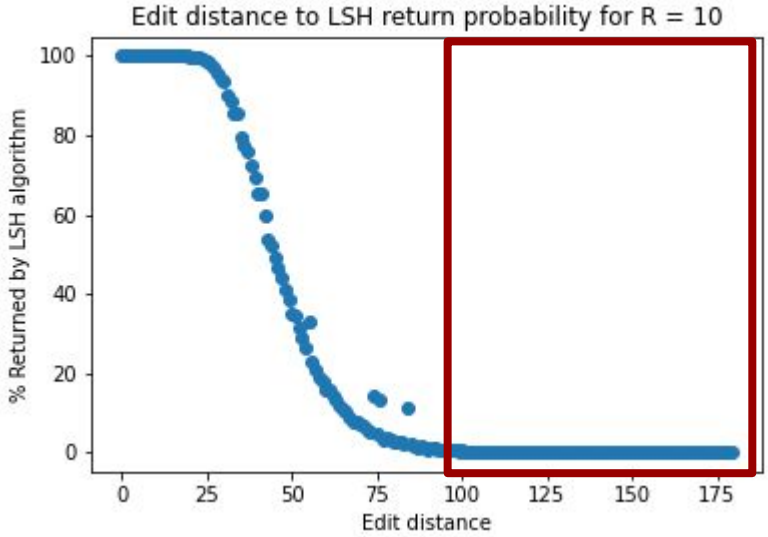
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

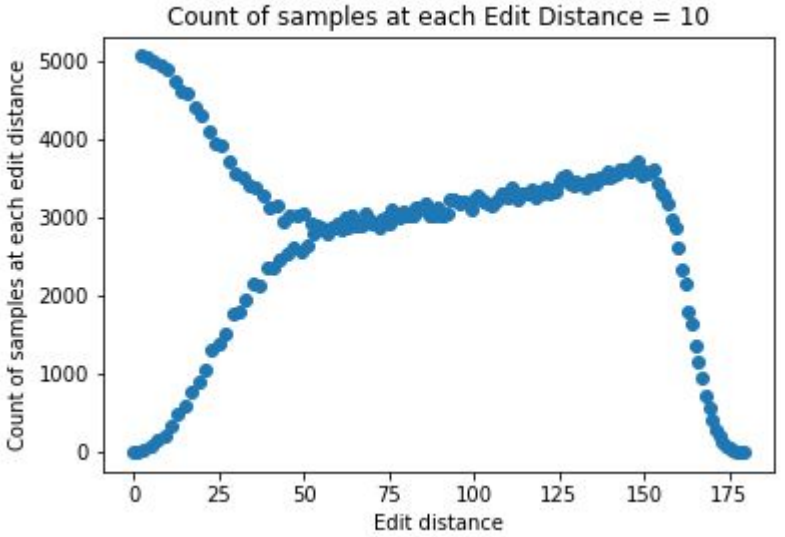



- When R = 10, the algorithm the probability of a sample being returned return of 100 > true_edit_distance > 25 , decreases exponentially as true_edit_distance rises

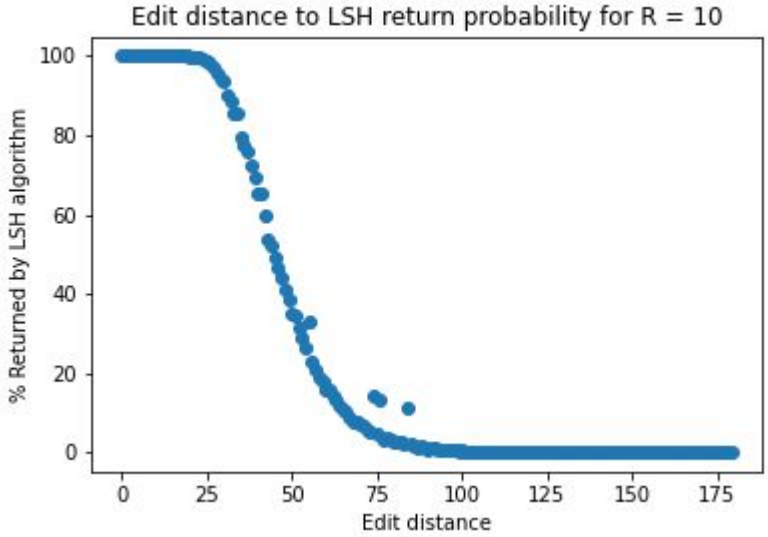
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

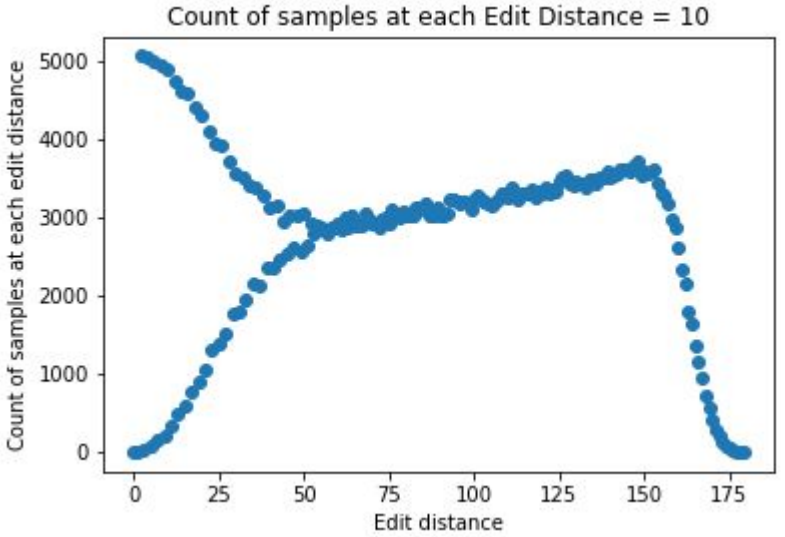



- When R = 10, the probability of a sample being returned of true_edit_distance > 100 was 0 across the 10 experiments

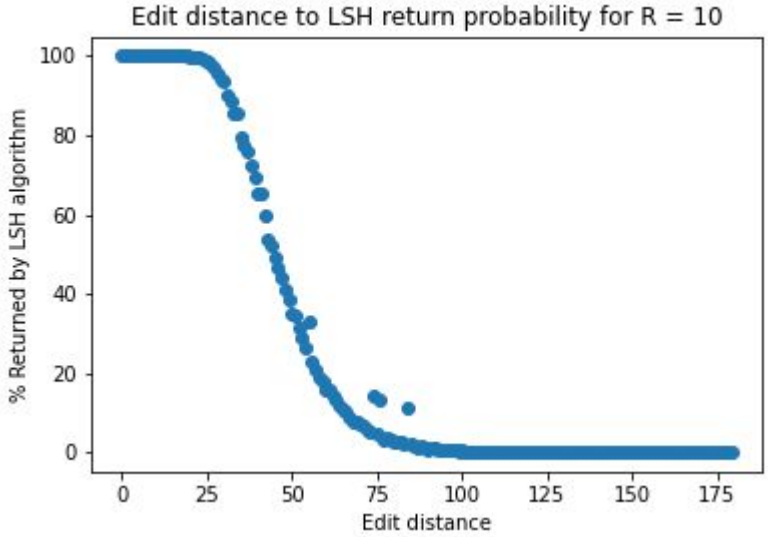
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

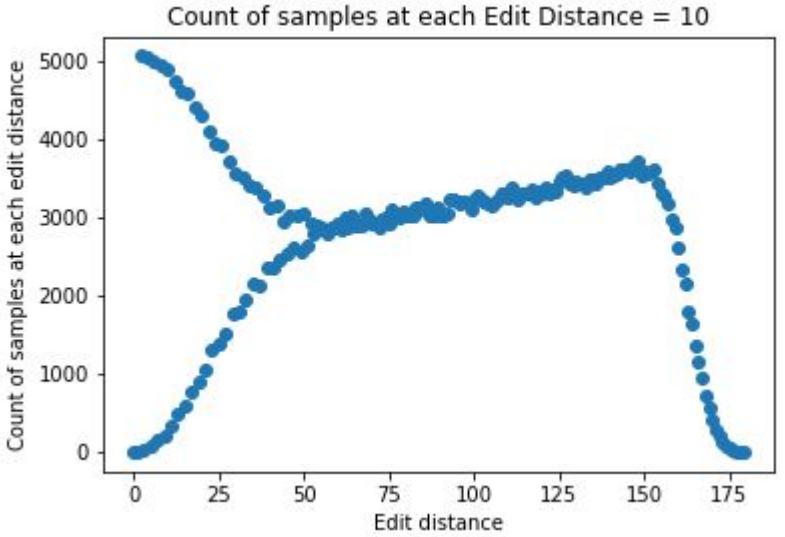



- Recall similar sample edit distance Function has the following properties:
  - As Z -> Infinity, the range [Z, 2Z] stays the same but the distribution becomes far more smooth

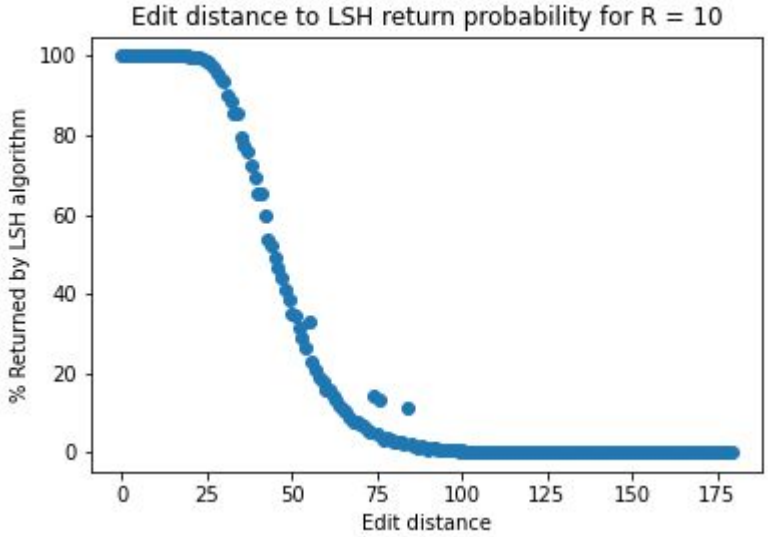
# Sampling Technique 3: Centered Random Sampling - Data R = 10

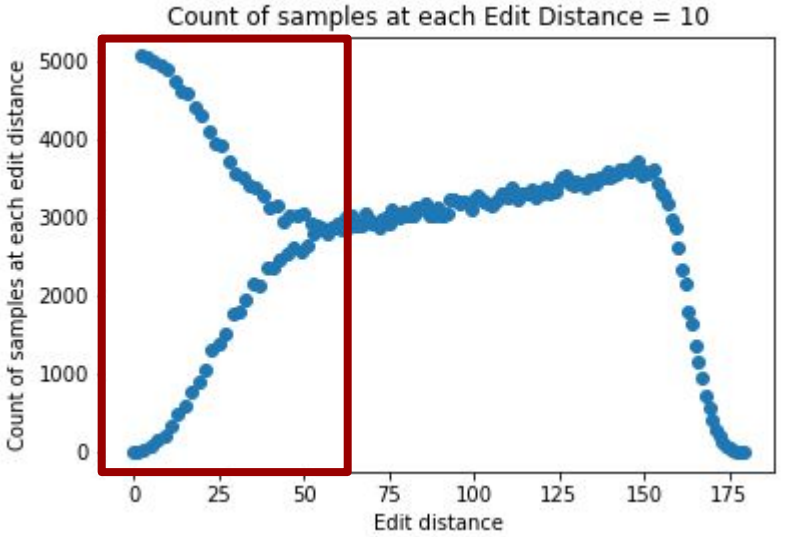



- Recall similar sample edit distance Function has the following property:
  - Returned string will have edit distance(ed) between [Z, 2Z] with PR(ed = 2Z) => 1 as Z -> 1 from above

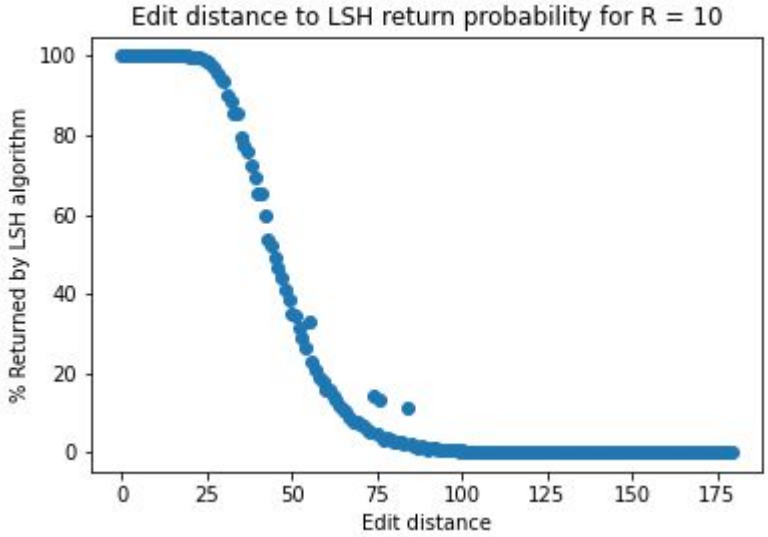
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

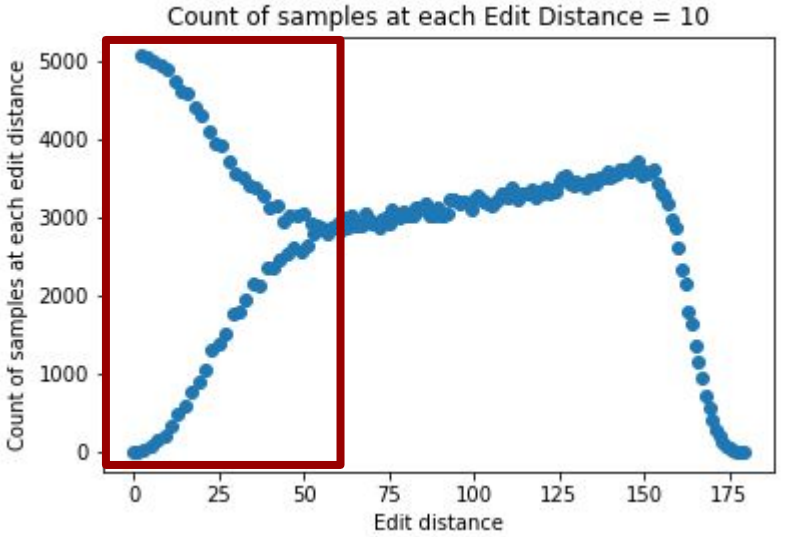



- This oscillating behavior at lower edit distances happens **simply because 2N is even** and PR(edit_distance_returned) -> 2Z as Z -> 1 from above.

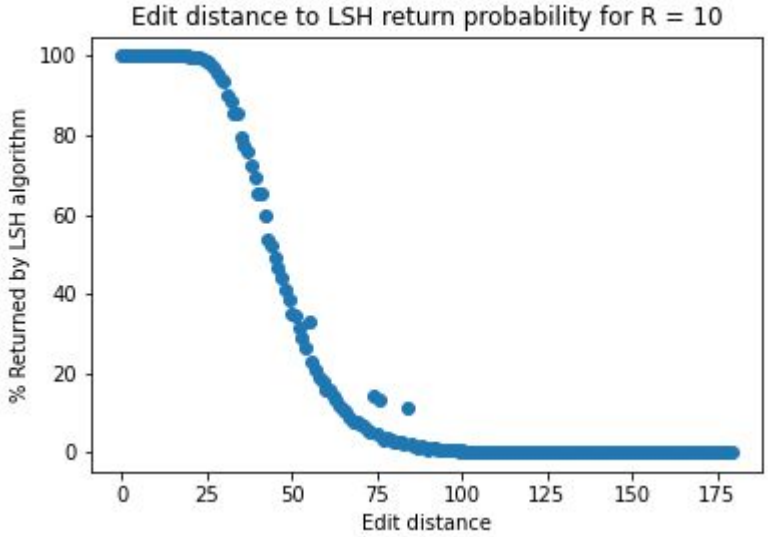
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

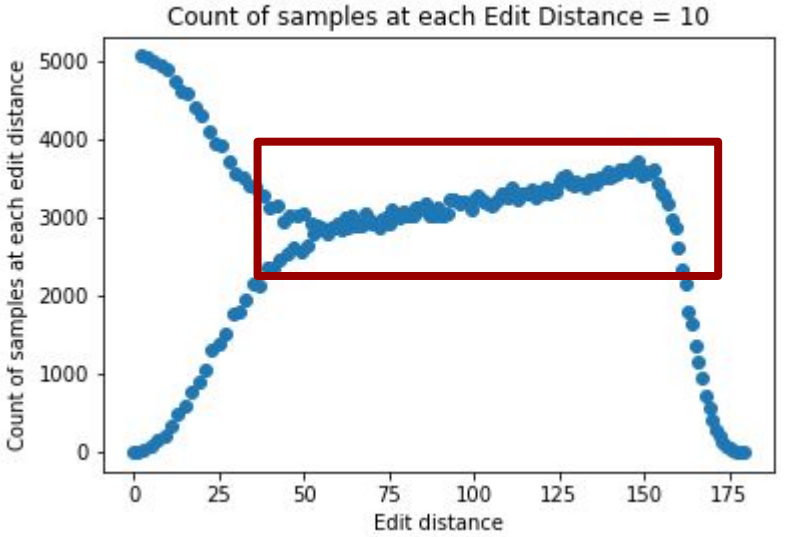



- Notice that if you were to ‘smooth’ the output by a taking the mean over a sliding window of length 2, **then the curve would be perfectly consistent with the rest of the plot**

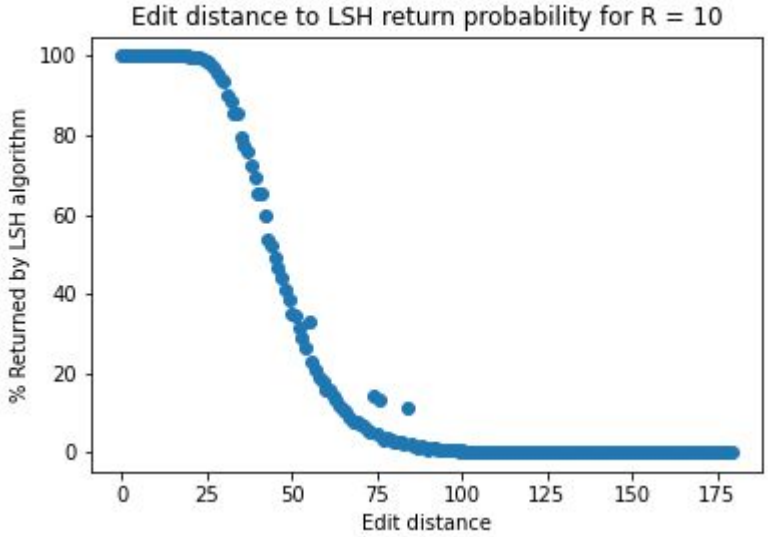
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

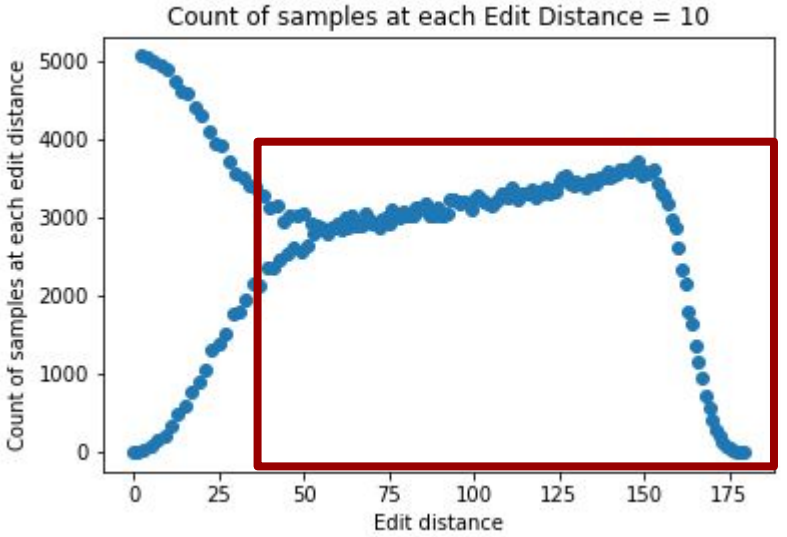



- Notice that if you were to ‘smooth’ the output by a taking the mean over a sliding window of length 2, **then the curve would be perfectly consistent with the rest of the plot**

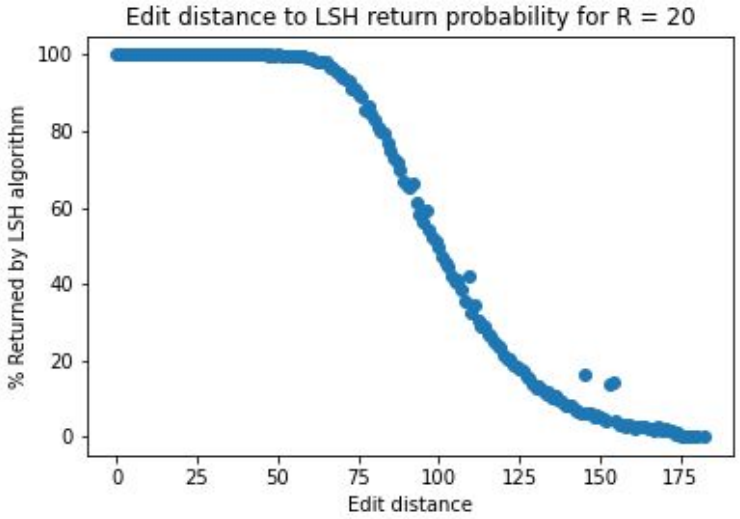
# Sampling Technique 3:  Centered Random Sampling - Data R = 10

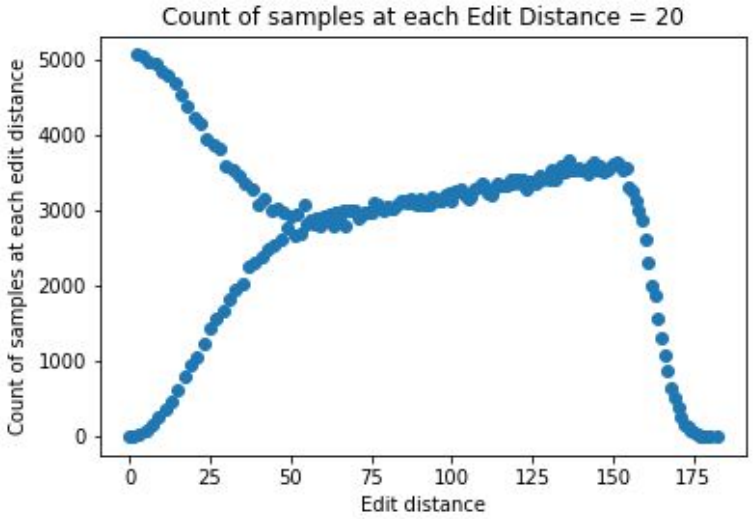



- No full parameter study was performed for similar sample edit distance function on edit distance based on N, Z.

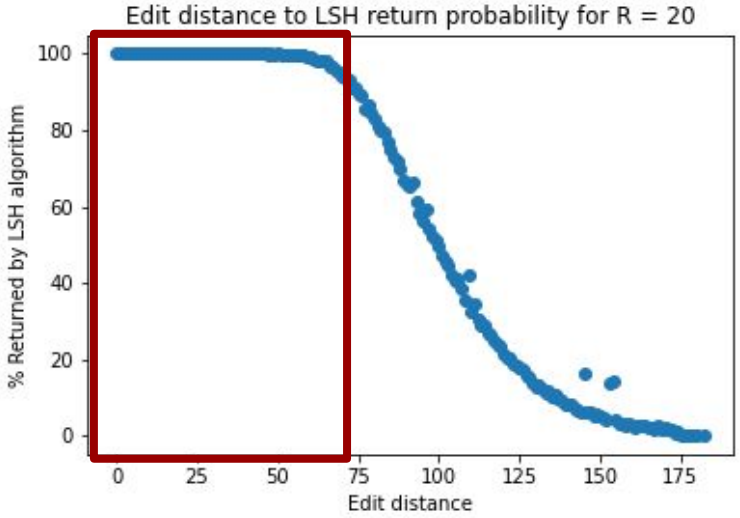
# Sampling Technique 3:  Centered Random Sampling - Data R = 0

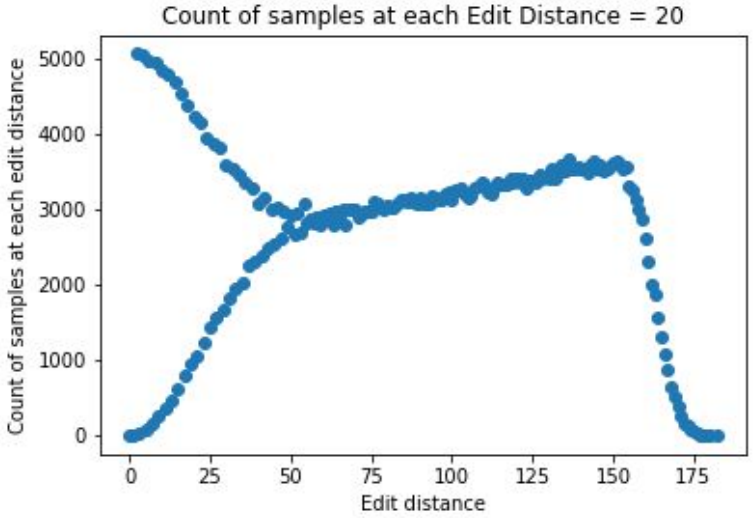



- For R = 20, % Returned by LSH algorithm on the left shows the exact same S curve behavior

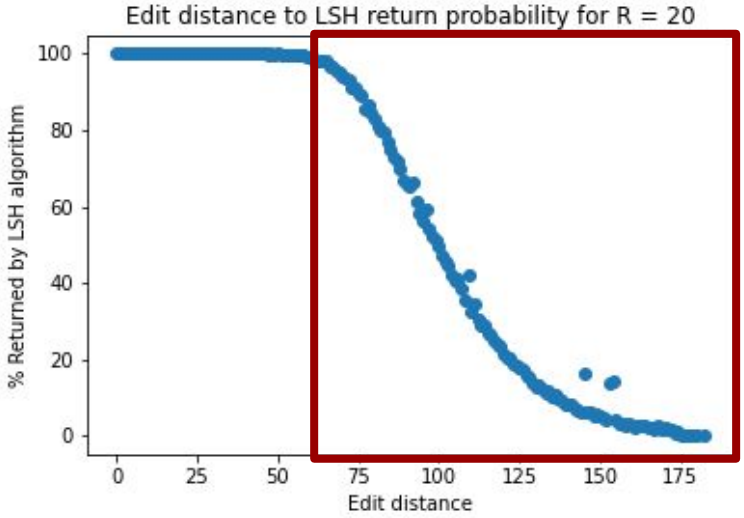
# Sampling Technique 3:  Centered Random Sampling - Data R = 20

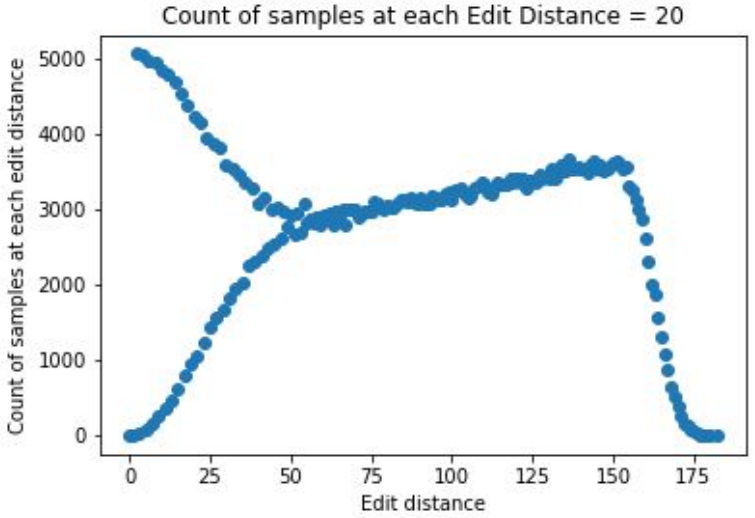



- For true_edit distance ~= 70, 100% of all matches were all returned

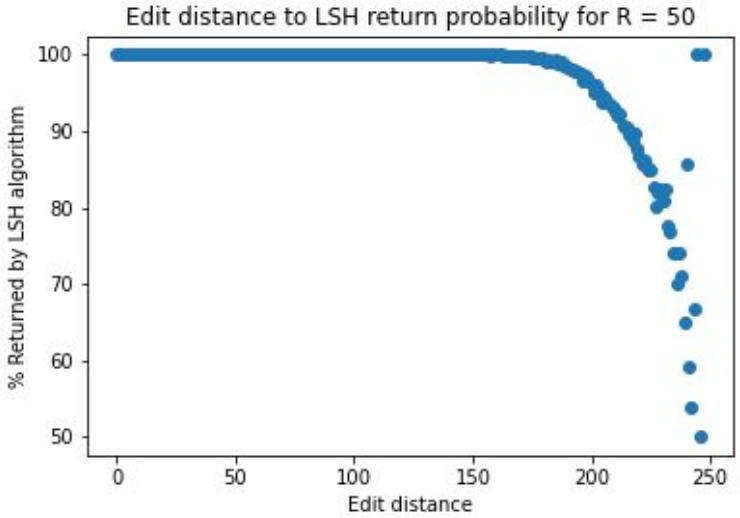
# Sampling Technique 3:  Centered Random Sampling - Data R = 20

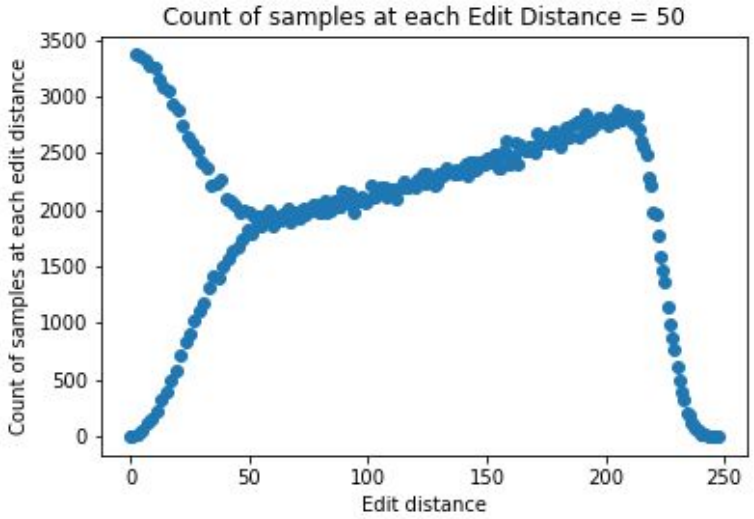



- % Returned by LSH shows exponential decay for 70 < true_edit_distance < 175

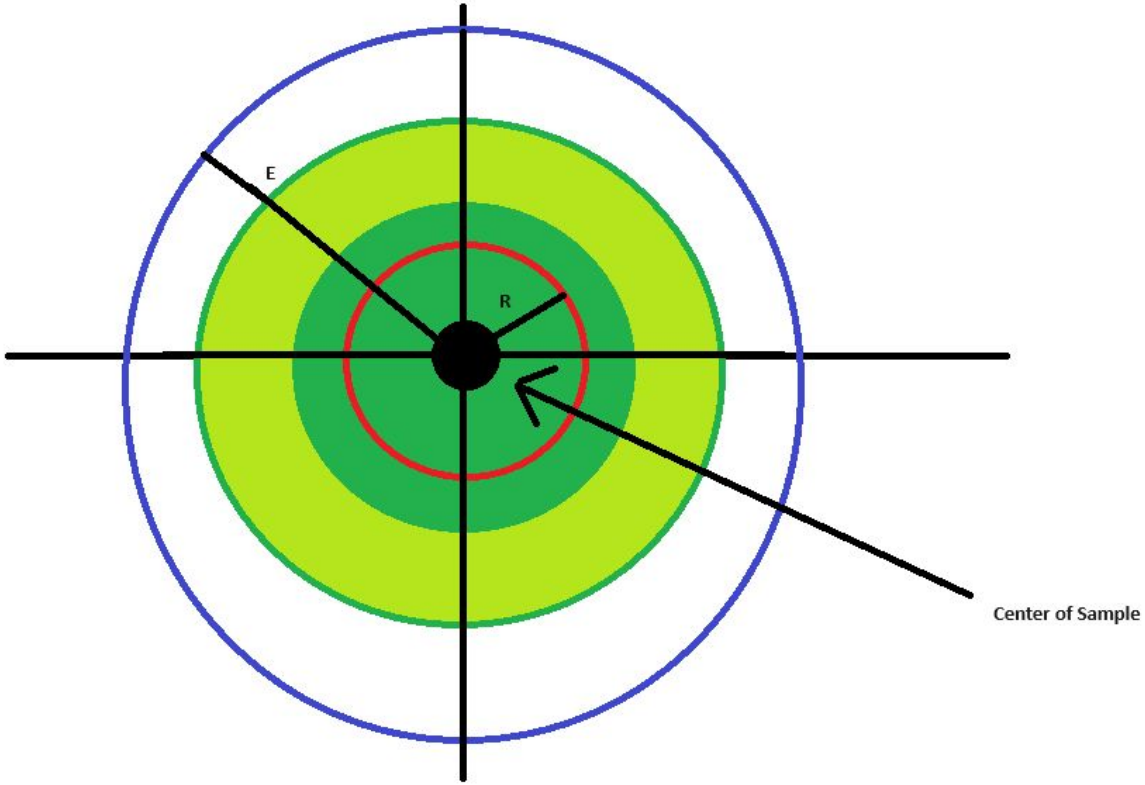
# Sampling Technique 3: Centered Random Sampling - Data R = 50




- For R = 50, notice the precision of the query drops dramatically.
- **R=50 case is still functionally relevant in a more randomized dataset due to the curse of dimensionality discussed previously**
- Recall for s1, s2, ***chosen at random*** from DNA(N) , the probability their edit distances range from [500, 540] is **extremely** high

# Sampling Technique 3:  Centered Random Sampling - 2D Visualization




- **Proven**:  The LSH algorithm returns all points within the dark green circle with 100% probability, and within the light green circle with between 0 and 100% probability.
- **Proven:**  The size of the light and dark green areas each grow with R

# Sampling Technique 3: Centered Random Sampling - Additional Discussion / Curse of Dimensionality

- The visualization for the return of the LSH algorithm given a query is only proven for a sample. Further, it is **very** likely that the size of the dark green and light green circles grows with R.
- However, for real world samples, this algorithm can be used as an engine to prove that two given DNA samples do not suffer from the curse of dimensionality in terms of edit distance. For practical use cases, this may be very helpful in terms of finding DNA matches in large databases.
    - Consider that for the curse of dimensionality can be proven that for the universe DNA(1000), 2 DNA strings will have an expected edit distance between 500 and 540. This algorithm, with the right configuration parameters, can be used to be proven beyond any doubt that two samples are outside the curse of dimensionality distance from one another.
    - For DNA(1000), S (longest common substring) falls in range between 8 and 12
    - For R = 50; S = floor(N/R+1) = 19.
    - **The algorithm can be used to prove a hypothesis test beyond any possible doubt.**

# Sampling Technique 3:  Centered Random Sampling - Additional Discussion / Hypothesis Test

- Given two random DNA strings, s1, s2, each of length N:
  - H0:  s1 and s2 have an edit distance between them at least equal to the minimum range of the curse of dimensionality distances
  - H1:  s1 and s2 have an edit distance between them no greater / far less than the minimum range of the curse of dimensionality distances - in other words, they have a statistically significant edit distance.
- For R=50, N=1000, all DNA samples fall under H1 as proven by previous slides.
- Discussion for Future Work:  If two DNA strings s1, s2, have a statistically significant edit distance, what else does that mean?

# Future Work - Theoretical

- Given two random DNA sequences each of length n, denoted s1, s2, who have a longest common substring length of S and an edit distance R:
  - What is Pr(R | N, S)?
    - Note:  This work is based on min(R|N,S)
  - What is Pr(S | N, R)
- How does this relate to the curse of dimensionality?
  - Large, random, samples converge to uniform ranges of R and S based on N.
  - What is P(R|N)?
  - What is P(S|N)?
    - ***Notice that P(R|N) and P(S|N) are related***.