CSE/STAT 416

Local Methods
Locality Sensitive Hashing

Pemi Nguyen
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 18, 2022

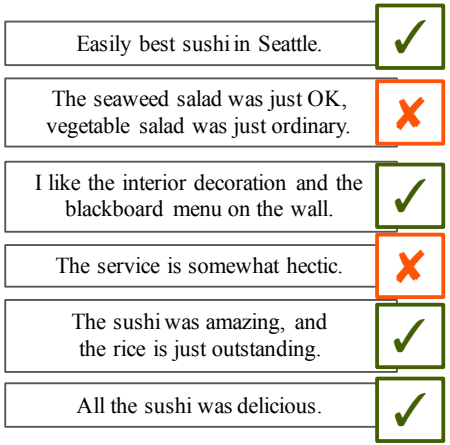
Pre-Lecture Video

Precision

What fraction of the examples I predicted positive were correct?

Sentences predicted to be positive:

$\hat{y}_i = +1$

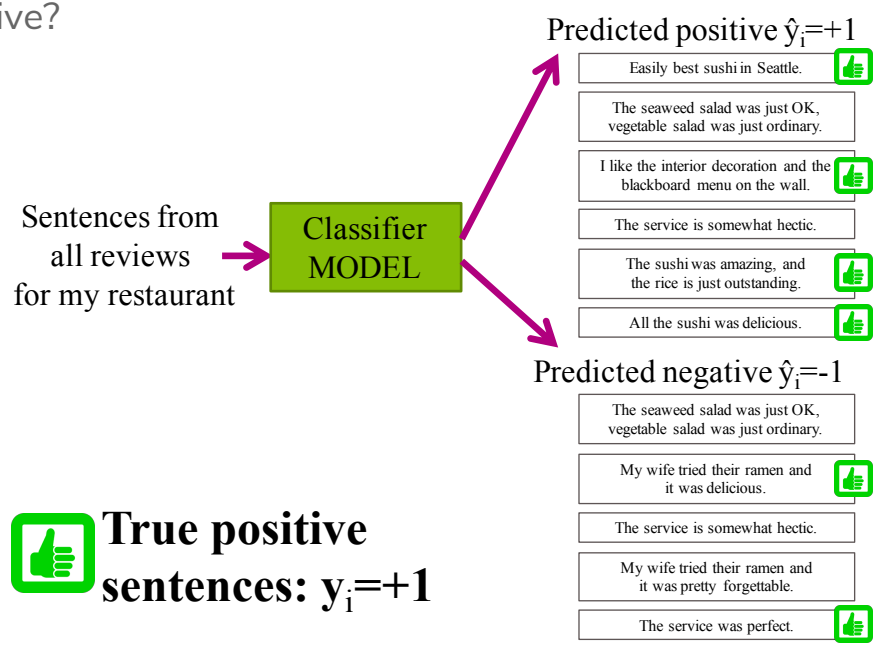


Only 4 out of 6 sentences predicted to be positive are actually positive

$$precision = \frac{C_{TP}}{C_{TP} + C_{FP}}$$

Recall

Of the truly positive examples, how many were predicted positive?



$$recall = \frac{C_{TP}}{N_P} = \frac{C_{TP}}{C_{TP} + C_{FN}}$$

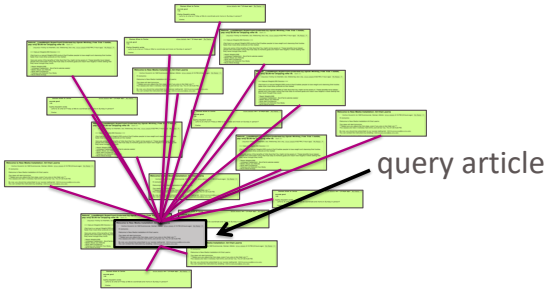
Document Retrieval

Consider you had some time to read a book and wanted to find other books similar to that one.

If we wanted to write an system to recommend books

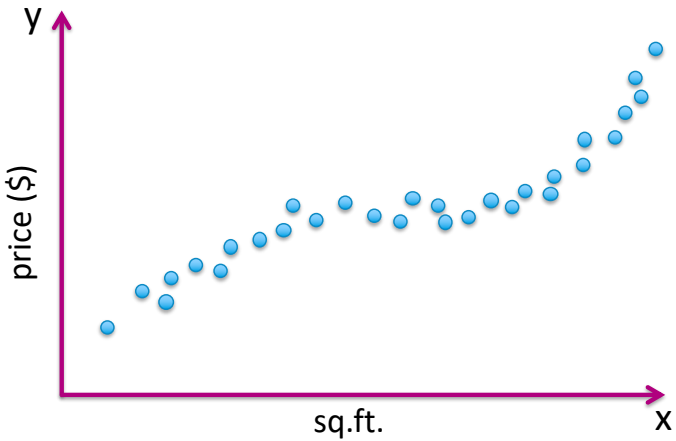
- How do we measure similarity?
- How do we search over books?
- How do we measure accuracy?

Big Idea: Define an **embedding** and a **similarity metric** for the books, and find the **"nearest neighbor"** to some query book.



Predicting House Prices

When we saw regression before, we assumed there was some linear/polynomial function that produced the data. All we had to do was choose the right polynomial degree.

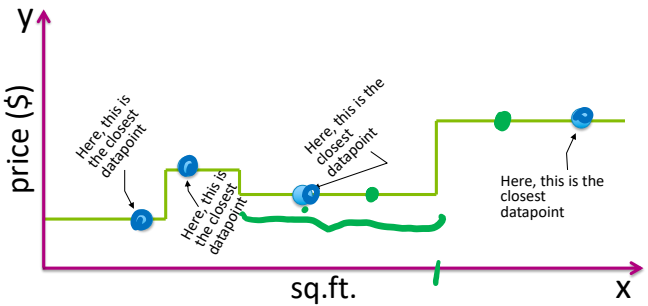


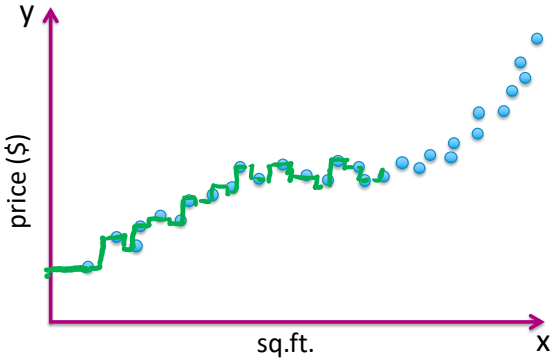
Predicting House Prices

What if instead, we didn't try to find the global structure, but instead just tried to infer values using local information instead.

Big Idea: Use 1-nearest neighbor to predict the price of a house.

Not actually a crazy idea, something realtors do sometimes!





1-NN Regression

Input: Query point: x_q, Training Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

$(x^{NN}, y^{NN}) = 1NearestNeighbor(x_q, \mathcal{D})$

Output: y^{NN}

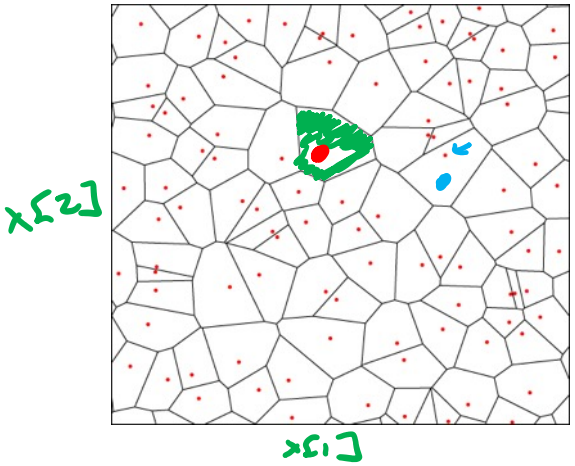
Where $1NearestNeighbor$ is the algorithm described yesterday to find the single nearest neighbor of a point.

Visualizing 1-NN Regression

The function learned by 1-NN is “locally constant” in each region nearest to each training point.

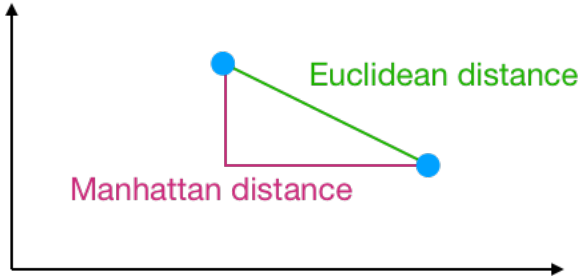
Can visualize this with a Voronoi Tessellation

- Shows all of the points that are “closest” to a particular training point
- Not actually computed in practice, but helps understand



Visualizing 1-NN Regression

Like last time, how you define “closest” changes predictions



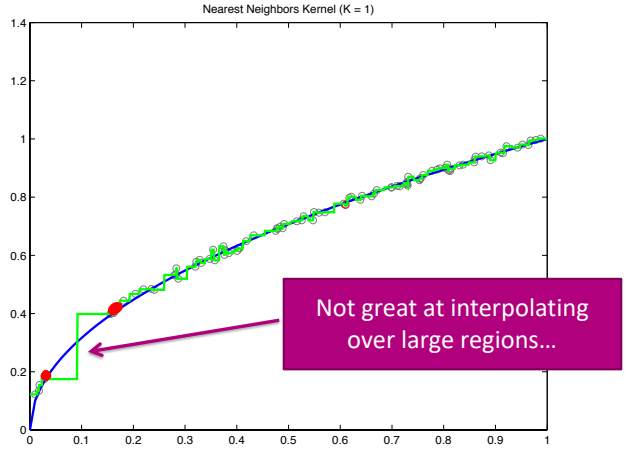
Euclidean distance

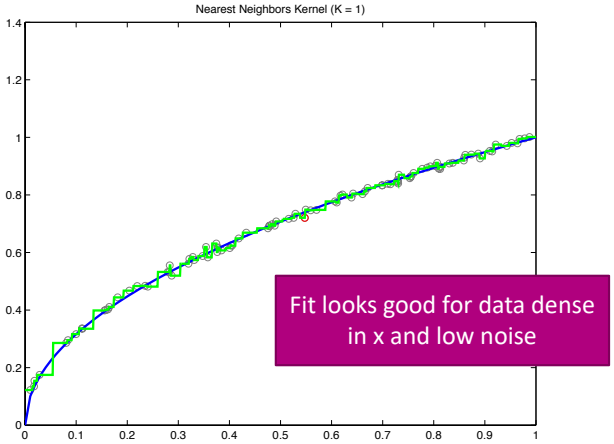
Manhattan distance

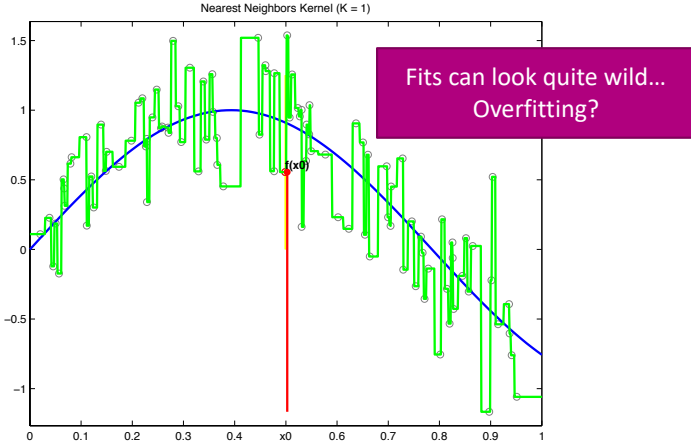
1-NN Regression

Weaknesses

- Inaccurate if data is sparse
- Can wildly overfit



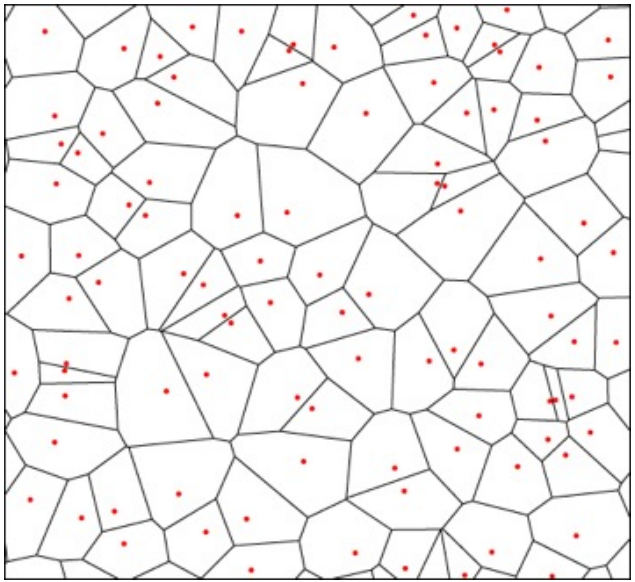




1-NN Classification

Can we use the same algorithm for classification? Yes!

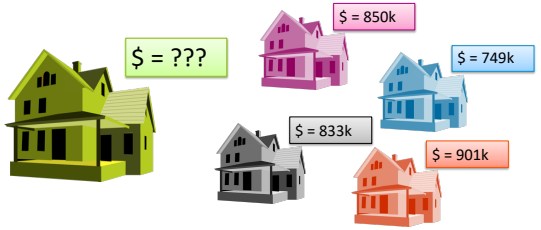
Predict the class of the nearest neighbor. Besides that, exactly the same as regression.



Prevent Overfitting

The downfalls of 1-NN come from it relies too much on a single data point (the nearest neighbor), which makes it susceptible to noise in the data.

More reliable estimate if you look at more than one house!

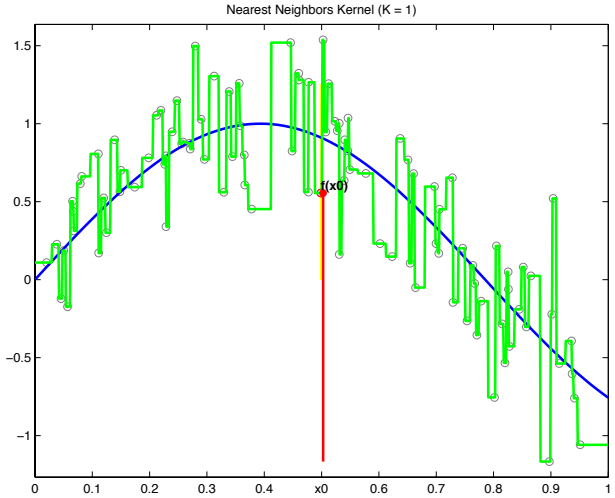


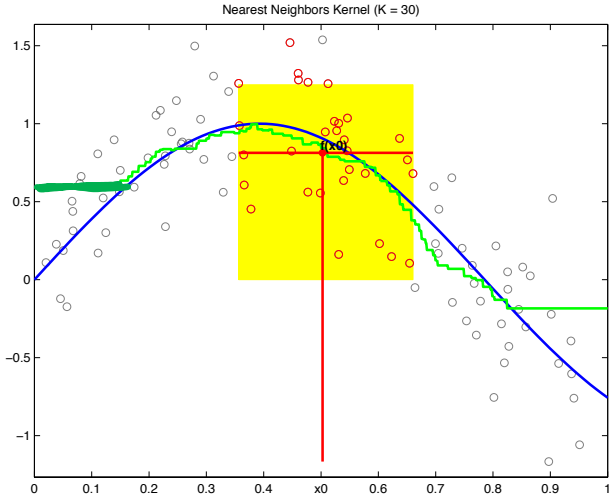
Input: Query point: x_q, Training Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

$(x^{NN_1}, y^{NN_1}), \ldots, (x^{NN_k}, y^{NN_k}) = kNearestNeighbor(x_q, \mathcal{D}, k)$

Output: $\hat{y}_q = \frac{1}{k}\sum_{j=1}^{k} y^{NN_j}$

k-NN Regression





By using a larger k, we make the function a bit less crazy

Still discontinuous though (neighbor is either in or out)

Boundaries are still sort of a problem

Issues with k-NN

While k-NN can solve some issues that 1-NN has, it brings some more to the table.

Have to choose right value of k.

- If k is too large, model is too simple

Discontinuities matter in many applications

- The error might be good, but would you believe a price jump for a 2640 sq.ft. house to a 2641 sq.ft. house?

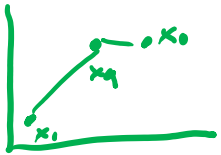
Seems to do worse at the boundaries still

Weighted k-NN

c_{q,NN_j} = weight between x_q and x^{NN_j}

Big Idea: Instead of treating each neighbor equally, put more weight on closer neighbors.

Predict:



$$\hat{y}_q = \frac{\sum_{j=1}^{k} c_{q,NN_j} y^{NN_j}}{\sum_{j=1}^{k} c_{q,NN_j}}$$

Reads: Weight each nearest neighbor by some value c_{q,NN_j}

How to choose c_{q,NN_j}?

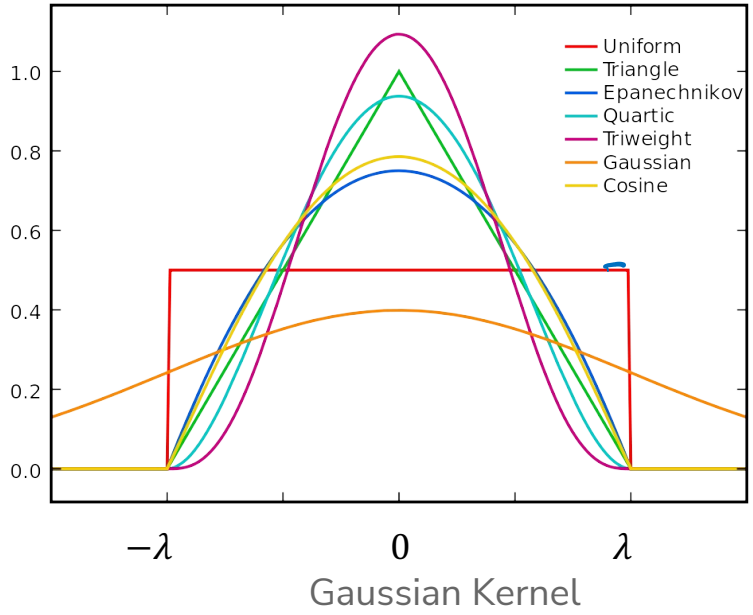
Want c_{q,NN_j} to be **small** if $dist\left(x_q, x^{NN_j}\right)$ is **large.**

Want c_{q,NN_j} to be **large** if $dist\left(x_q, x^{NN_j}\right)$ is **small.**

(Optional) Kernels

Use a function called a **kernel** to turn distance into weight that satisfies the properties we listed before.

$$c_{q,NN_j} = Kernel_\lambda(dist(x_q, x^{NN_j}))$$



Gaussian Kernel

$$Kernel_\lambda\left(dist(x_i, x_q)\right) = \exp\left(-\frac{dist(x_i, x_q)^2}{\lambda}\right)$$

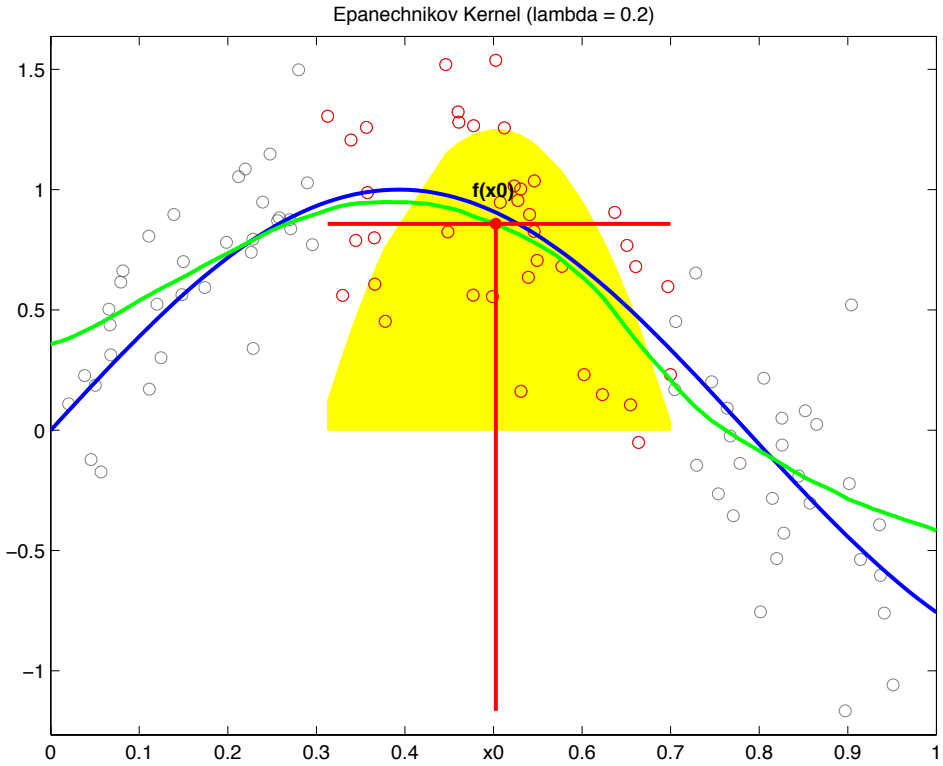
(Optional) Kernel Regression

We can take this one step farther, instead of just using a kernel to weight the k nearest neighbors, can use the kernel to weight **all training points**! This is called **kernel regression**.

$$\hat{y}_q = \frac{\sum_{i=1}^{n} c_{q,i} y_i}{\sum_{i=1}^{n} c_{q,i}} = \frac{\sum_{i=1}^{n} Kernel_\lambda\left(dist(x_i, x_q)\right) y_i}{\sum_{i=1}^{n} Kernel_\lambda\left(dist(x_i, x_q)\right)}$$

(Optional) Visualizing Kernel Regression

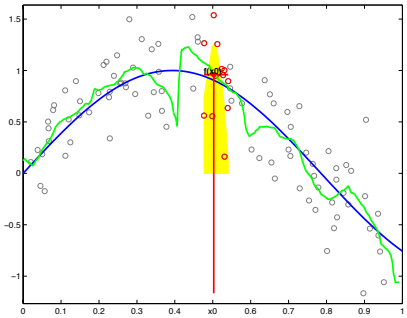
This kernel has bounded support, only look at values $\pm\lambda$ away



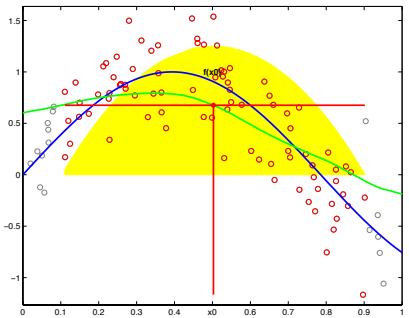
(Optional) Choose Bandwidth λ

Often, which kernel you use matters much less than which value you use for the bandwidth λ

λ = 0.04



λ = 0.4



How to choose? Cross validation or a validation set to choose

- Kernel
- Bandwidth
- K (if using weighted k-NN, not needed for kernel regression)

Poll Everywhere

Think

1.5 min

pollev.com/cs416

In a few sentences, compare and contrast the following ML models.

- k-Nearest Neighbor Regression
- Weighted k-Nearest Neighbor Regression
- Kernel Regression

Poll Everywhere

Think

3 min

pollev.com/cs416

In a few sentences, compare and contrast the following ML models.

- k-Nearest Neighbor Regression
- Weighted k-Nearest Neighbor Regression
- Kernel Regression

Efficient Nearest Neighbors

Nearest Neighbor Efficiency

Nearest neighbor methods are good because they require no training time (just store the data, compute NNs when predicting).

How slow can that be? Very slow if there is a lot of data!

- $\mathcal{O}(n)$ if there are n data points.
- If n is in the hundreds of billions, this will take a while...

There is not an obvious way of speeding this up unfortunately.

Big Idea: Sacrifice accuracy for speed. We will look for an approximate nearest neighbor to return results faster

Approximate Nearest Neighbor

Don’t find the exact NN, find one that is “close enough”.

Many applications are okay with approximate answers

- The measure of similarity is not perfect
- Clients probably can’t tell the difference between the most similar book and a book that’s pretty similar.

We will use **locality sensitive hashing** to answer this approximate nearest neighbor problem.

High level approach

- Design an algorithm that yields a close neighbor with high probability
- These algorithms usually come with a “guarantee” of what probability they will succeed, won’t discuss that in detail but is important when making a new approximation algorithm.

Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing is an algorithm that answers the approximate nearest neighbor problem.

Big Idea

> Break the data into smaller bins based on how close they are to each other
>
> When you want to find a nearest neighbor, choose an appropriate bin and do an exact nearest neighbor search for the points in that bin.

More bins → Fewer points per bin → Faster search

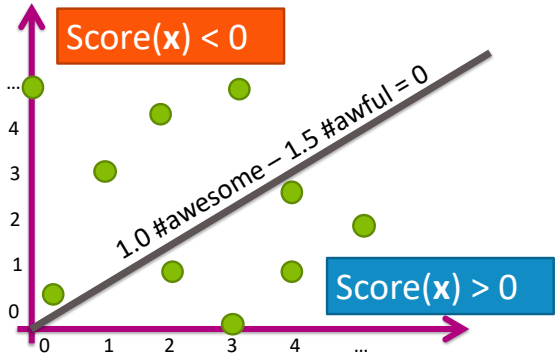
More bin → More likely to make errors if we aren't careful

Binning

How do we make the bins?

What if we pick some line that separates the data and then put them into bins based on the $Score(x)$ for that line?

Looks like classification, but we don't have labelled data here. Will explain shortly how to find this line.

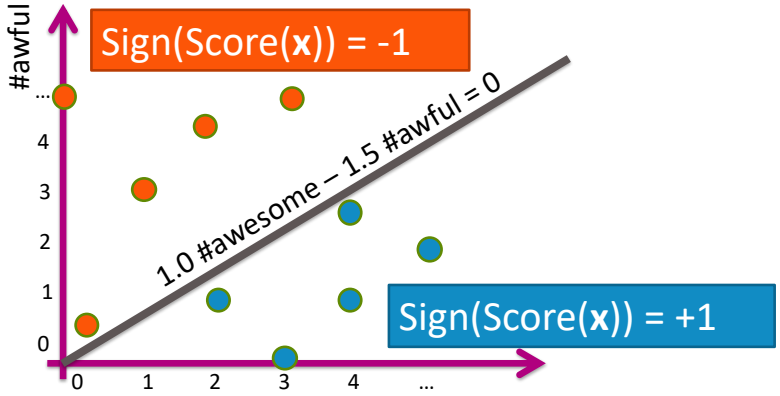


Binning

Put the data in bins based on the sign of the score (2 bins total)

Call negative score points bin 0, and the other bin 1 (**bin index**)

2D Data	Sign(Score)
x_1 = [0, 5]	-1
x_2 = [1, 3]	-1
x_3 = [3, 0]	1
...	...



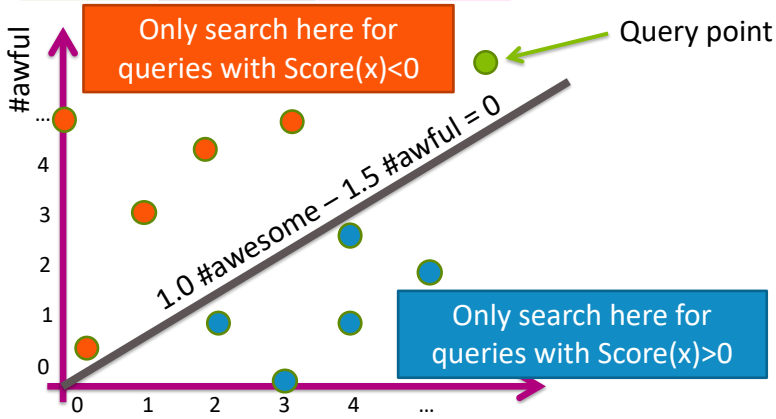
Binning

Put the data in bins based on the sign of the score (2 bins total)

Call negative score points bin 0, and the other bin 1 (**bin index**)

2D Data	Sign(Score)	Bin index
x_1 = [0, 5]	-1	0
x_2 = [1, 3]	-1	0
x_3 = [3, 0]	1	1
...	...	...

candidate neighbors if Score(x)<0



When asked to find neighbor for query point, only search through points in the same bin!

This reduces the search time to $\frac{n}{2}$ if we choose the line right.

LSH with 2 bins

Create a table of all data points and calculate their bin index based on some chosen line

2D Data	Sign(Score)	Bin index
$x_1 = [0, 5]$	-1	0
$x_2 = [1, 3]$	-1	0
$x_3 = [3, 0]$	1	1
…	…	…

Store it in a hash table for fast lookup

Bin	0	1
List containing indices of datapoints:	{1,2,4,7,…}	{3,5,6,8,…}

HASH TABLE

When searching for a point x_q:

- Find its bin index based on that line
- Search over the points in that bin

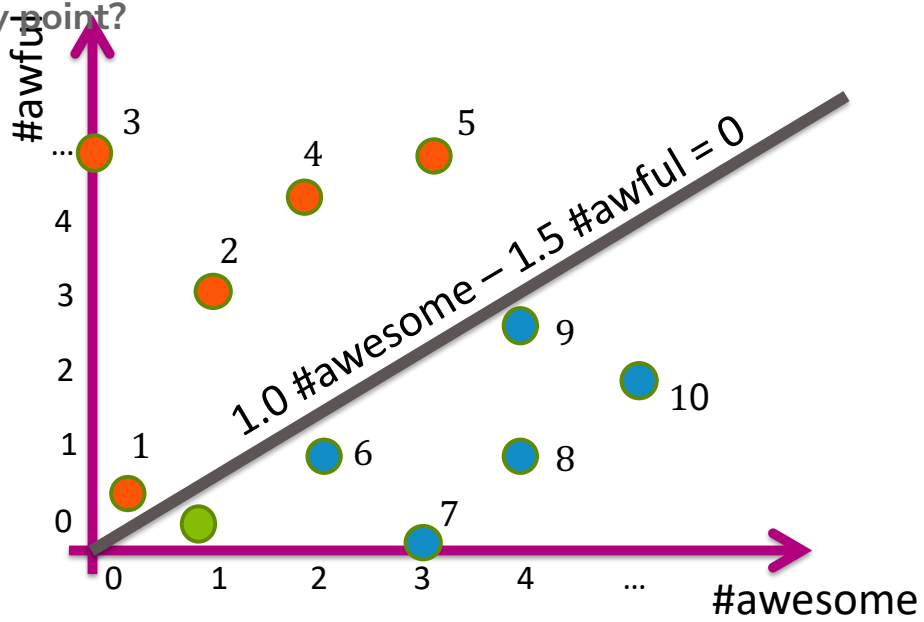
Poll Everywhere

Think

1 min

pollev.com/cs416

If we used LSH with this line, what would be the result returned for searching for the nearest neighbor of the green query point?



Some Issues

1. How do we find a good line that divides the data in half?
2. Potential Errors: Points close together might be split up into separate bins
3. Large computation cost: Only dividing the points in half doesn't speed things up that much...

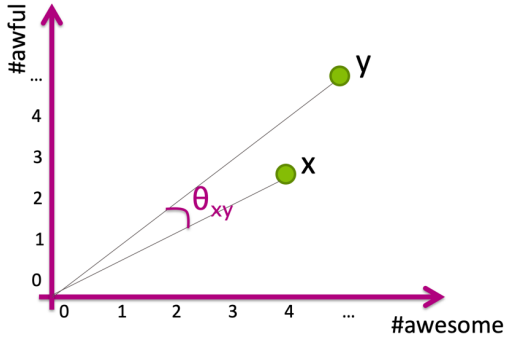
How to choose line?

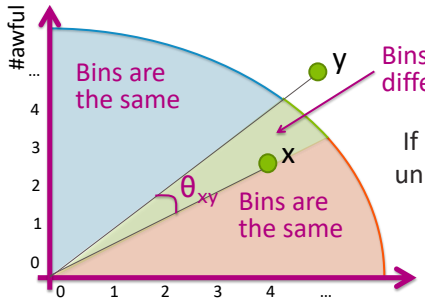
Wild Idea: Choose the line randomly!

Choose a slope randomly between 0 and 90 degrees

How bad can randomly picking it be?

If two points have a small cosine distance, it is unlikely that we will split them into different bins!





If θ_{xy} is small (x,y close), unlikely to be placed into separate bins

Some Issues

1. How do we find a good line that divides the data in half?
2. Potential Errors: Points close together might be split up into separate bins
3. Large computation cost: Only dividing the points in half doesn't speed things up that much...

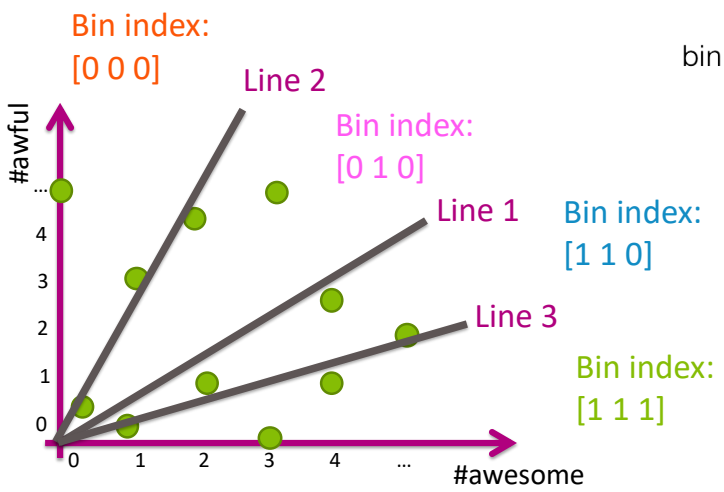
Brain Break

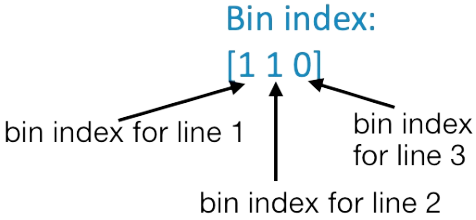
3:30

More Bins

Can reduce search cost by adding more lines, increasing the number of bins.

For example, if we use 3 lines, we can make more bins!





LSH with Many Bins

Create a table of all data points and calculate their bin index based on some chosen lines. Store points in hash table indexed by all bin indexes

2D Data	Sign ($Score_1$)	Bin 1 index	Sign ($Score_2$)	Bin 2 index	Sign ($Score_3$)	Bin 3 index
x_1 = [0, 5]	-1	0	-1	0	-1	0
x_2 = [1, 3]	-1	0	-1	0	-1	0
x_3 = [3, 0]	1	1	1	1	1	1
…	…	…	…	…	…	…

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

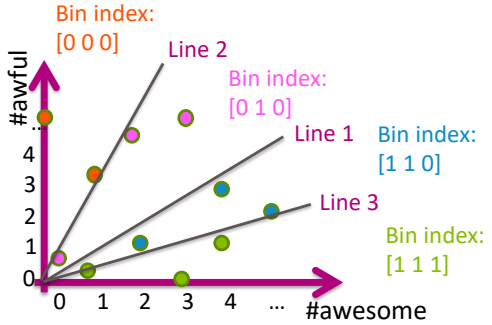
When searching for a point x_q:

- Find its bin index based on the lines
- Only search over the points in that bin

LSH Example

Imagine my query point was (2, 2)

This has bin index [0 1 0]



Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

By using multiple bins, we have reduced the search time!

However, it's more likely that we separate points from their true nearest neighbors since we do more splits ☹

- Often with approximate methods, there is a tradeoff between speed and accuracy.

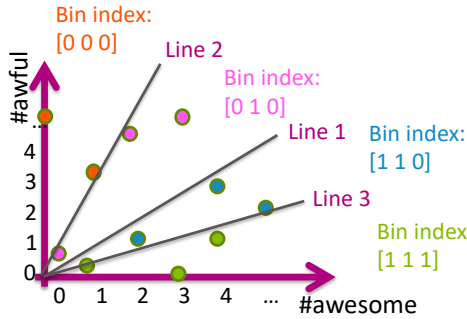
Improve Quality

The nice thing about LSH is we can actually tune this tradeoff by looking at nearby bins. If we spend longer searching, we are likely to find a better answer.

What does ”nearby” mean for bins?

Bin	[0 0 0] = 0	[0 0 1] = 1	[0 1 0] = 2	[0 1 1] = 3	[1 0 0] = 4	[1 0 1] = 5	[1 1 0] = 6	[1 1 1] = 7
Data indices:	{1,2}	--	{4,8,11}	--	--	--	{7,9,10}	{3,5,6}

Next closest bins (flip 1 bit)



In practice, set some time “budget” and keep searching nearby bins until budget runs out

Locality Sensitive Hashing (LSH)

Pre-Processing Algorithm

- Draw h lines randomly
- For each data point, compute $Score(x_i)$ for each line
- Translate the scores into binary indices
- Use binary indices as a key to store the point in a hash table

Querying LSH

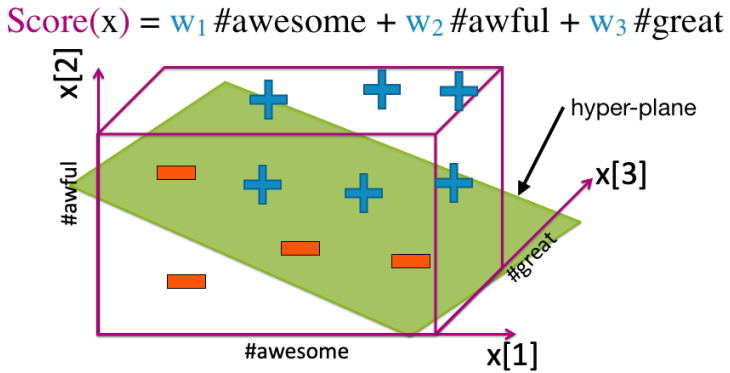
- For query point x_q compute $Score(x_q)$ for each of the h lines
- Translate scores into binary indices. Lookup all data points that have the same key.
- Do exact nearest neighbor search just on this bin.
- If there is more time in the computation budget, go look at nearby bins until this budget runs out.

Higher Dimensions

Pick random hyper-plane to separate points for points in higher dimensions.

Unclear how to pick h for LSH and you can't do cross-validation here (why?)

Generally people use $h \approx \log(d)$



Recap

Theme: Use local methods for classification and regression and speed up nearest neighbor search with approximation methods.

Ideas:

- 1-NN Regression and Classification
- k-NN Regression and Classification
- Weighted k-NN vs Kernel Regression
- Locality Sensitive Hashing