

Precision

What fraction of the examples I predicted positive were correct?

Sentences predicted to be positive:

$$\hat{y}_i = +1$$

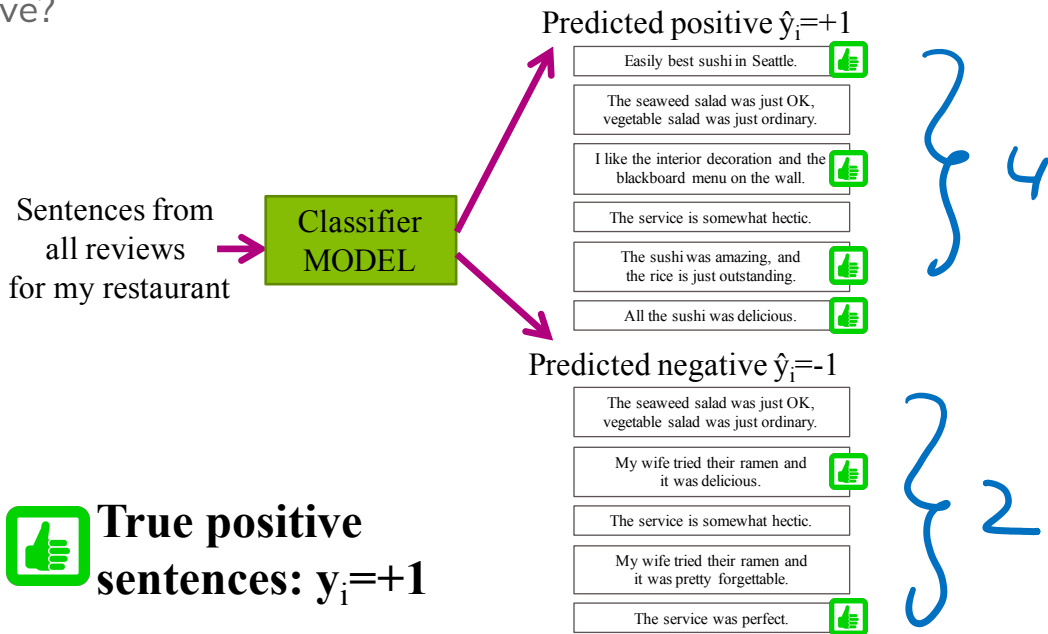
Easily best sushi in Seattle.	✓
The seaweed salad was just OK, vegetable salad was just ordinary.	✗
I like the interior decoration and the blackboard menu on the wall.	✓
The service is somewhat hectic.	✗
The sushi was amazing, and the rice is just outstanding.	✓
All the sushi was delicious.	✓

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

$$precision = \frac{C_{TP}}{C_{TP} + C_{FP}} = \frac{4}{4+2} = \frac{2}{3}$$

Recall

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



$$recall = \frac{C_{TP}}{N} = \frac{C_{TP}}{C_{TP} + C_{FN}} = \frac{4}{4+2} = \frac{2}{3}$$

Document Retrieval

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

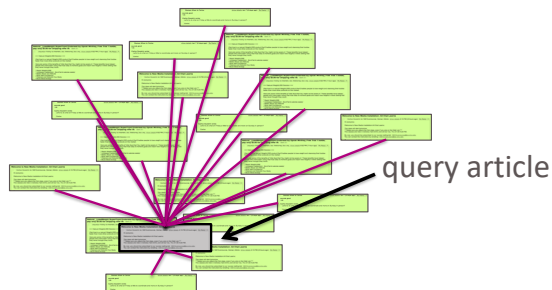
If we wanted to write a system to recommend books

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

*Bow
TF-IDF*

*euclidean dist
cosine dist*

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

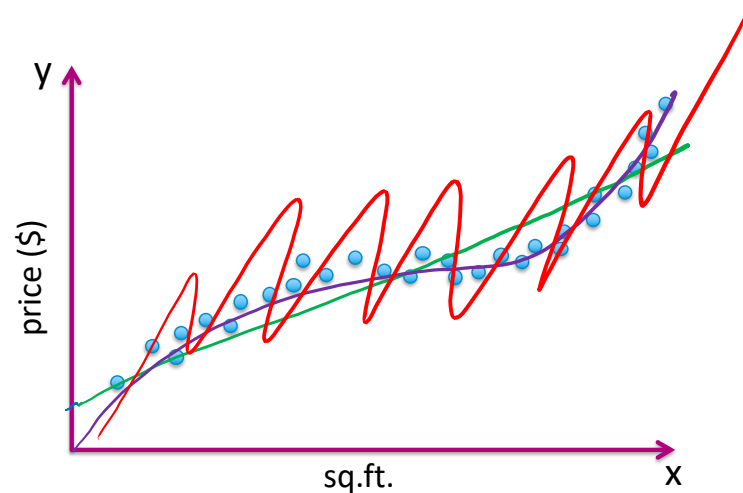


Predicting House Prices

$$y_i \approx f(x_i) + \epsilon_i$$

Global

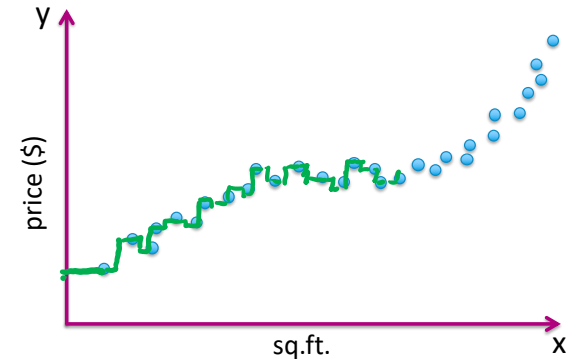
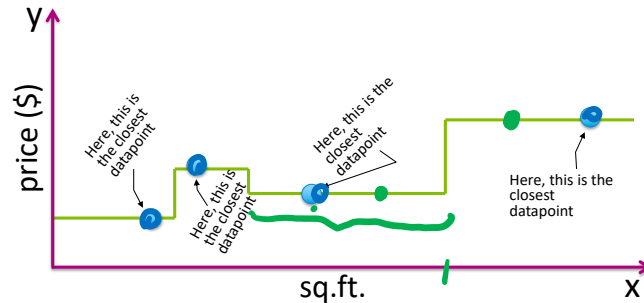
When we saw regression before, we assumed there was some linear/polynomial function that produced the data. All we had to do was choose the right polynomial degree.



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!



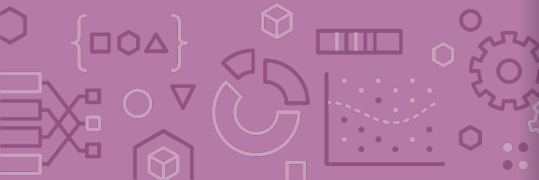
1-NN Regression

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.



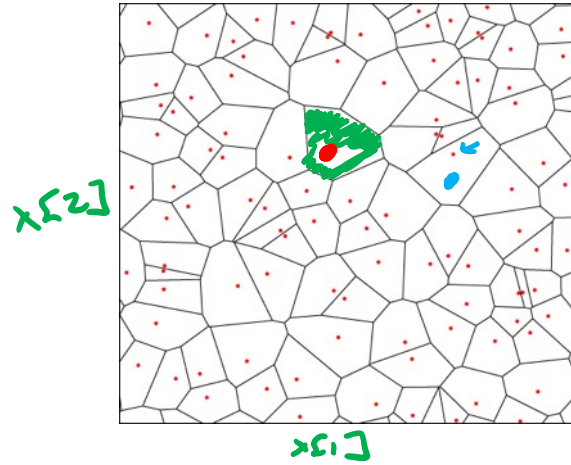
Visualizing 1-NN Regression

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

Can visualize this with a Voronoi Tessellation

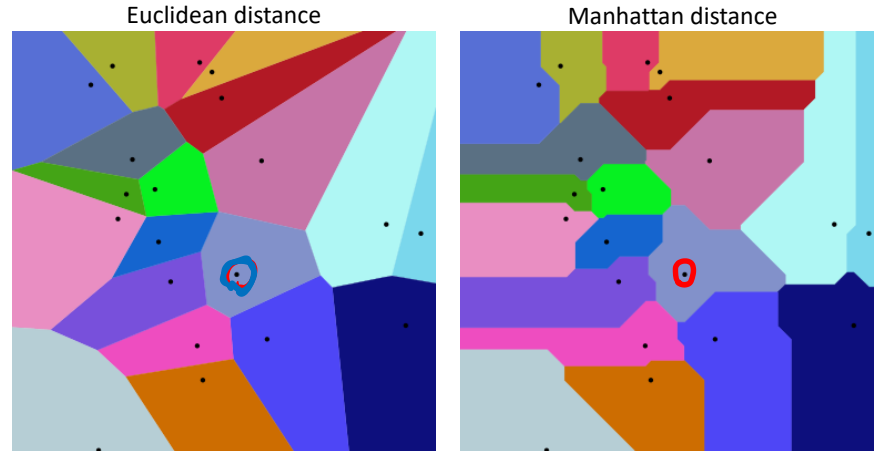
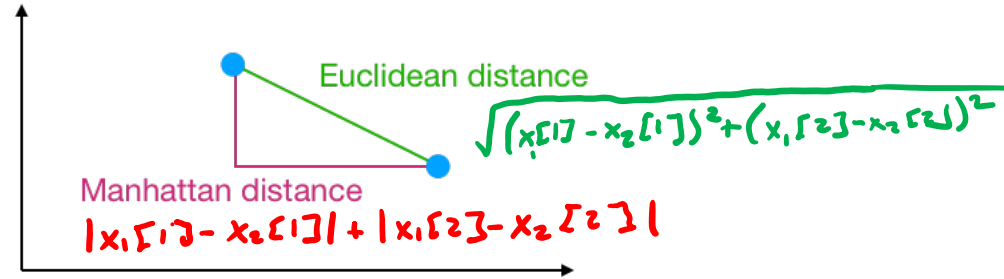
Shows all of the points that are “closest” to a particular training point

Not actually computed in practice, but helps understand



Visualizing 1-NN Regression

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

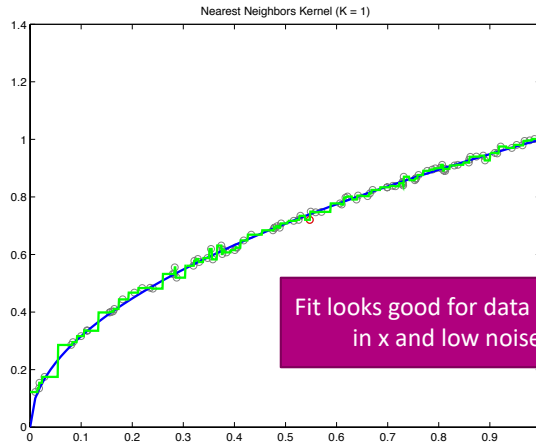


1-NN Regression

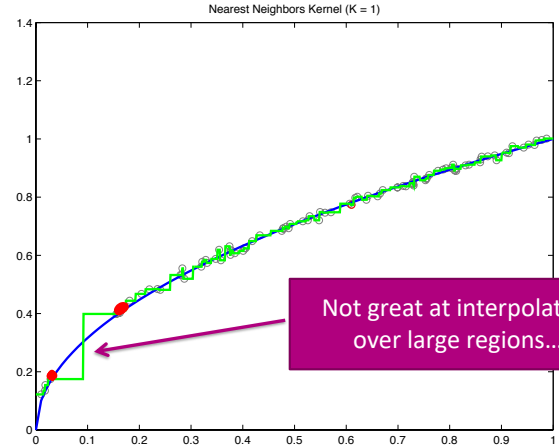
Weaknesses

Inaccurate if data is sparse

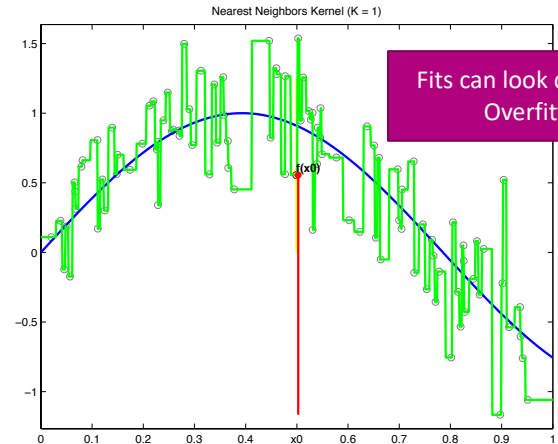
Can wildly overfit



Fit looks good for data dense
in x and low noise



Not great at interpolating
over large regions...

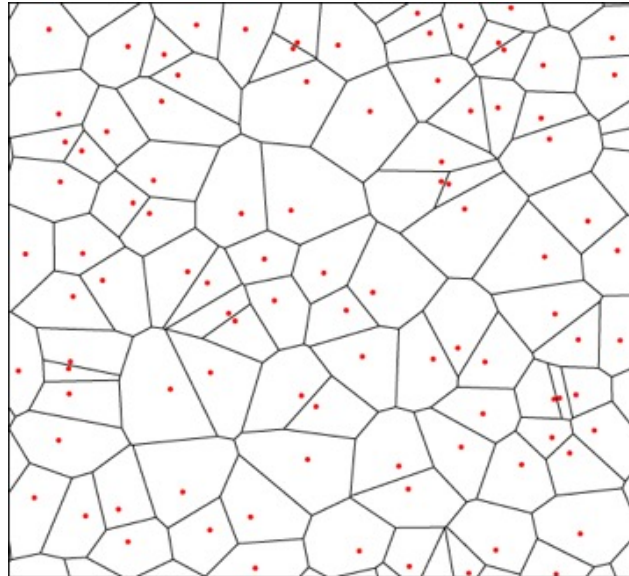


Fits can look quite wild...
Overfitting?

1-NN Classification

Can we use the same algorithm for classification? Yes!

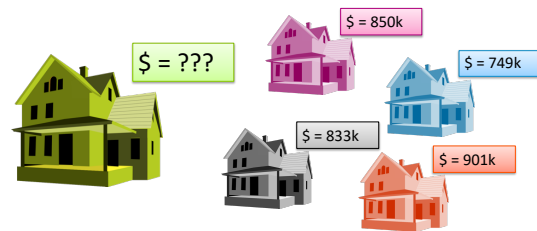
Predict the class of the nearest neighbor. Besides that, exactly the same as regression.



Prevent Overfitting

The downsides 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!

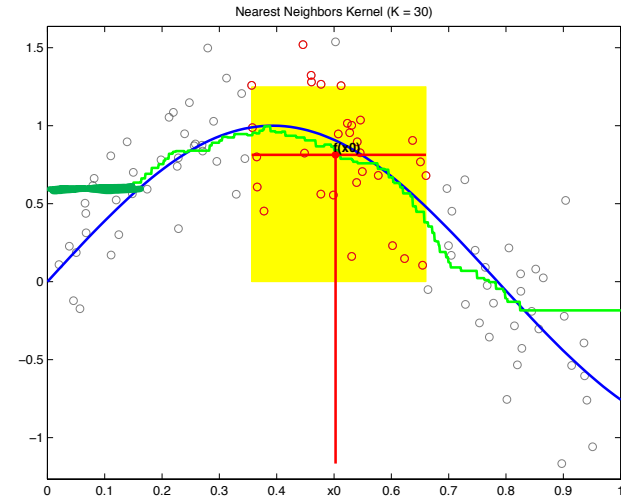
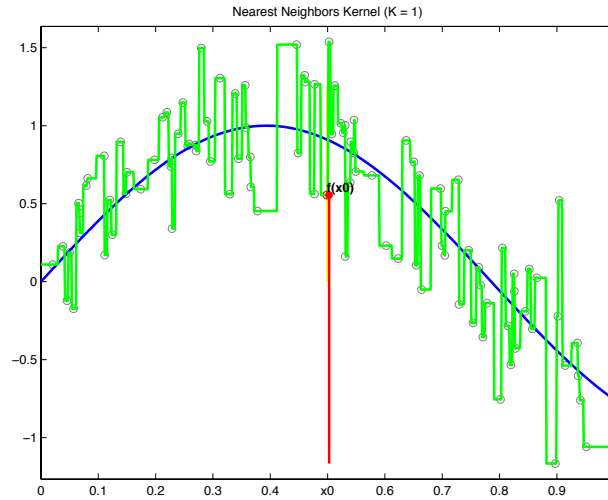


Input: Query point: x_q , Training Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

$(x^{NN_1}, y^{NN_1}), \dots, (x^{NN_k}, y^{NN_k}) = k\text{NearestNeighbor}(x_q, \mathcal{D}, k)$

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

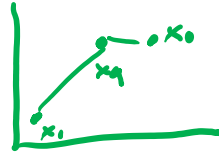


Weighted k-NN

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

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 $dist(x_q, x^{NN_j})$ is **large**.

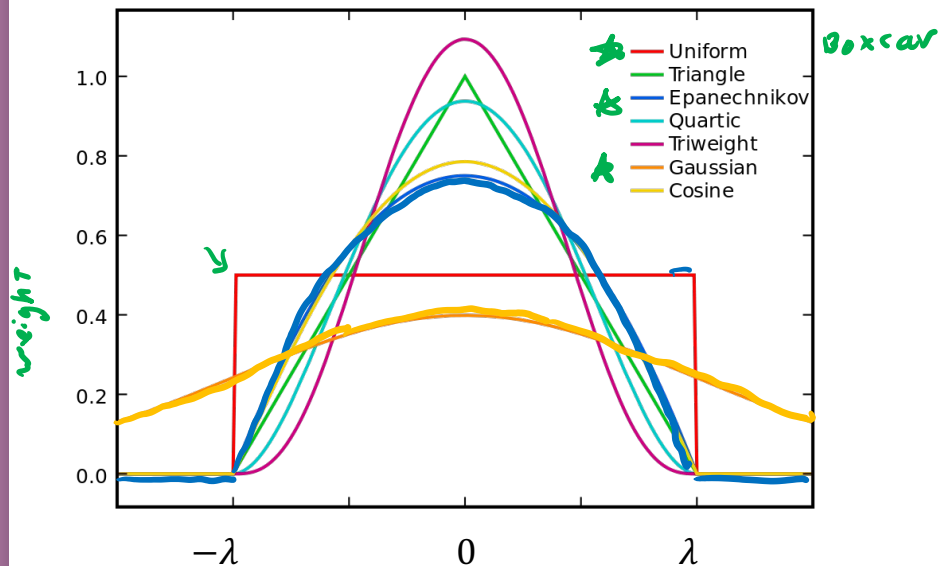
Want c_{q, NN_j} to be **large** if $dist(x_q, x^{NN_j})$ is **small**.

Kernels

$\lambda = \text{bandwidth}$

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

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



Gaussian Kernel

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

Kernel Regression

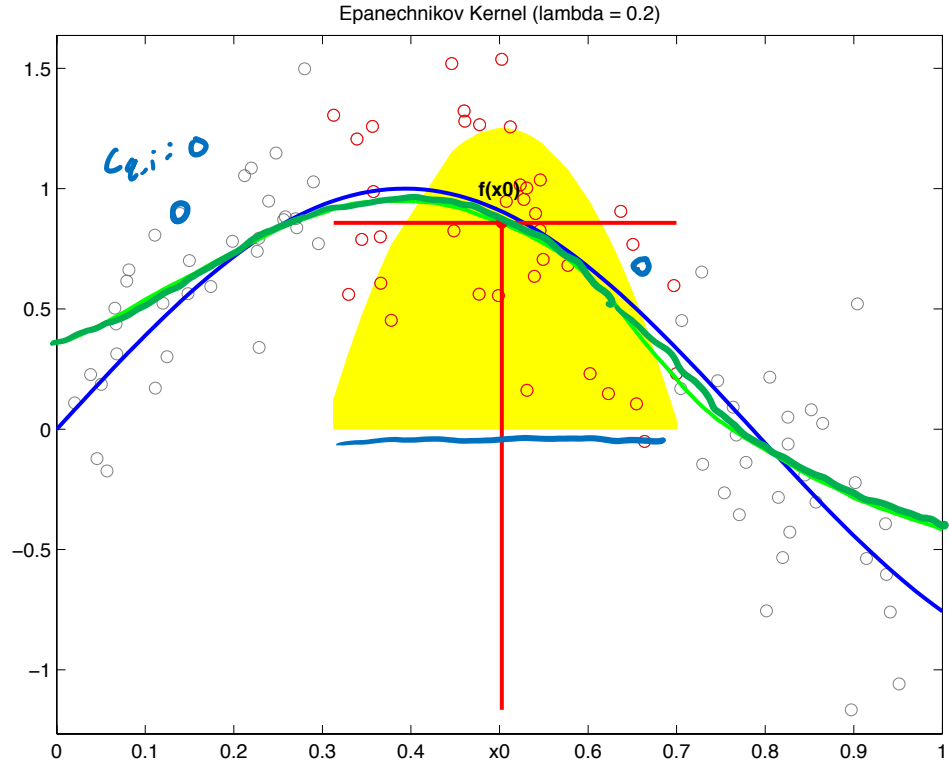
We can take this one step farther, instead of just using a kernel to weight the k nearest neighbors, can use the kernel to weight **all training points**! This is called **kernel regression**.

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



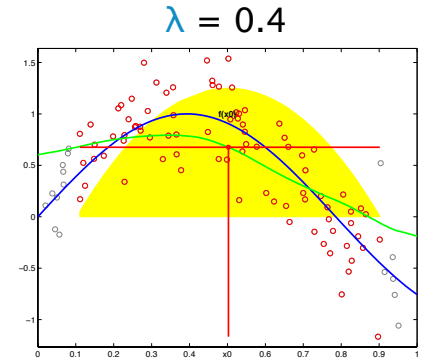
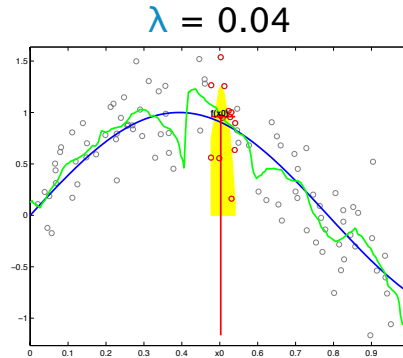
Visualizing Kernel Regression

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



Choose Bandwidth λ

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



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

Kernel

Bandwidth

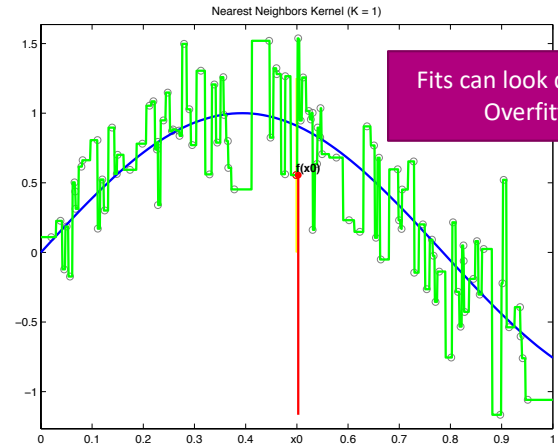
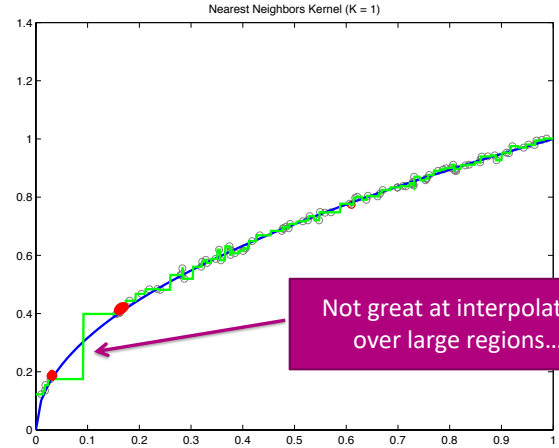
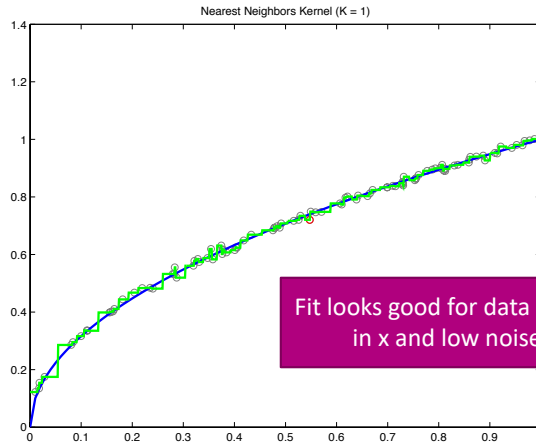
K (if using weighted k-NN, not needed for kernel regression)

1-NN Regression

Weaknesses

Inaccurate if data is sparse

Can wildly overfit

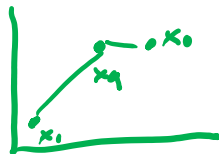


Weighted k-NN

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

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 $\text{dist}(x_q, x^{NN_j})$ is **large**.

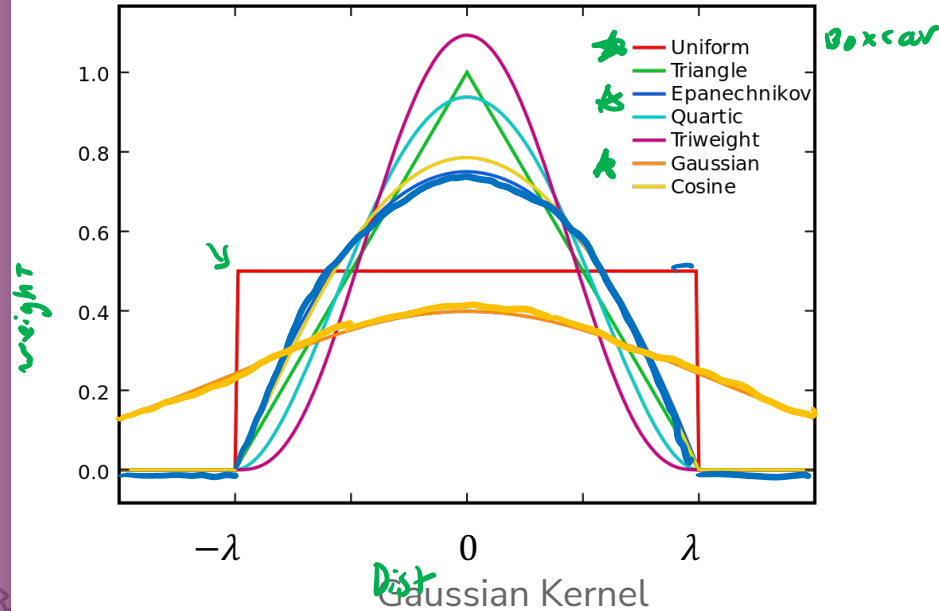
Want c_{q, NN_j} to be **large** if $\text{dist}(x_q, x^{NN_j})$ is **small**.

Kernels

$\lambda = \text{bandwidth}$

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

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

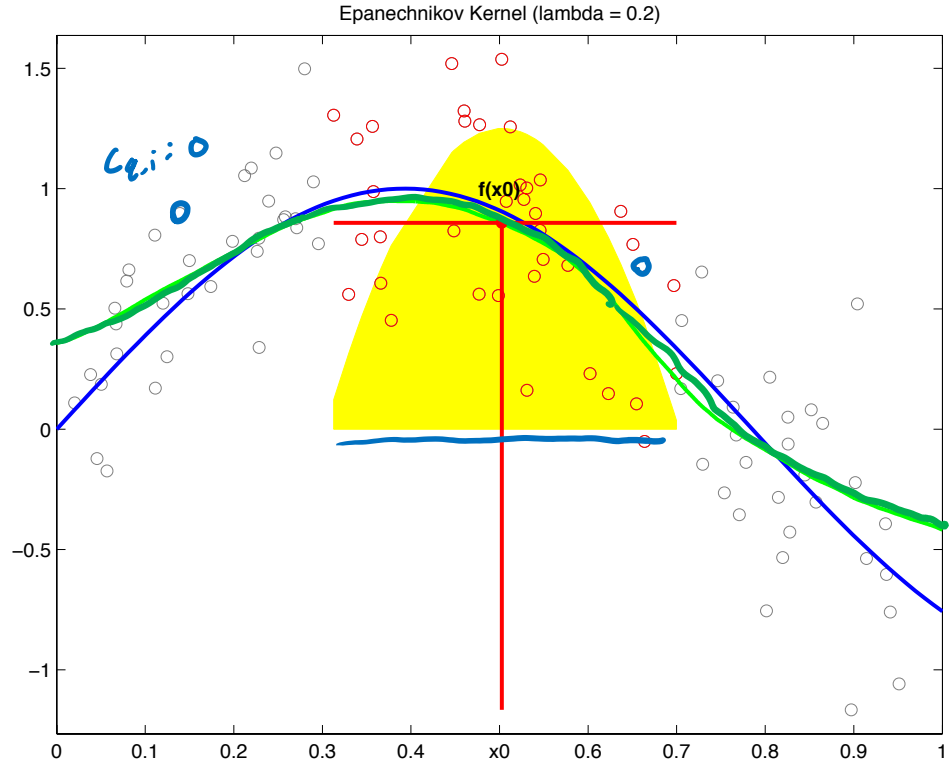


Gaussian Kernel

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

Visualizing Kernel Regression

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



slido

Think 

1.5 min

slido #cs416

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

k-Nearest Neighbor Regression

Weighted k-Nearest Neighbor Regression

Kernel Regression

slido

Group



3 min

slido #cs416

All: take averages

Non-parametric method

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

k-Nearest Neighbor Regression

• Weighting: Unweighted

• Data: k-NN

Weighted k-Nearest Neighbor Regression

• Weighting: Kernel

• Data: k-NN

Kernel Regression

• Weighting: Kernel

• Data: All data points

↳ May have weight 0

if kernel assigns weight 0

to dist $> \lambda$

Kernel regression + boxcar kernel

\approx k-NN

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).

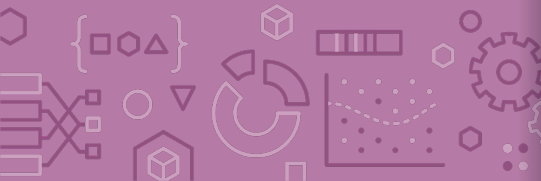
How slow can that be? Very slow if there is a lot of data!

$O(n)$ 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.

LSH

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

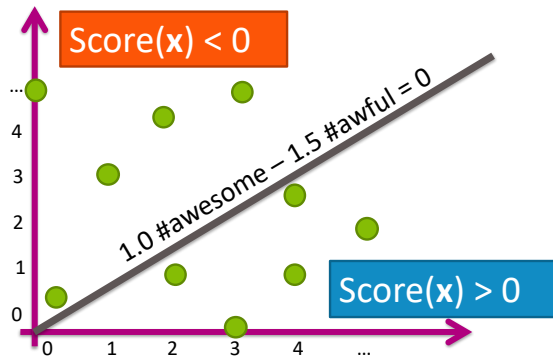


Binning

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.

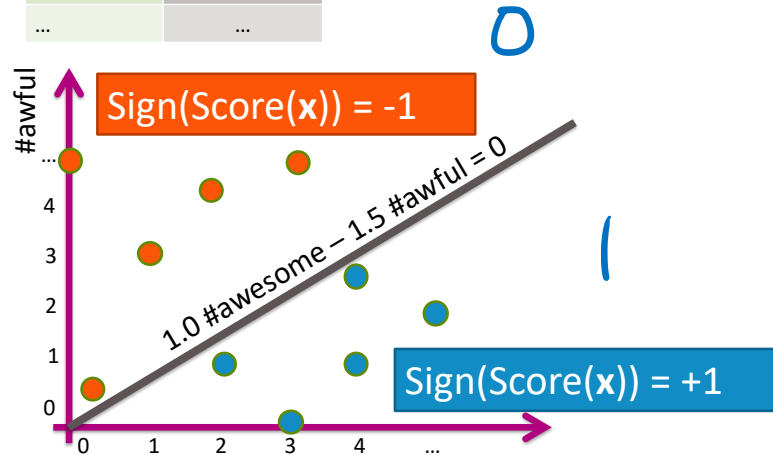


Binning

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
...	...



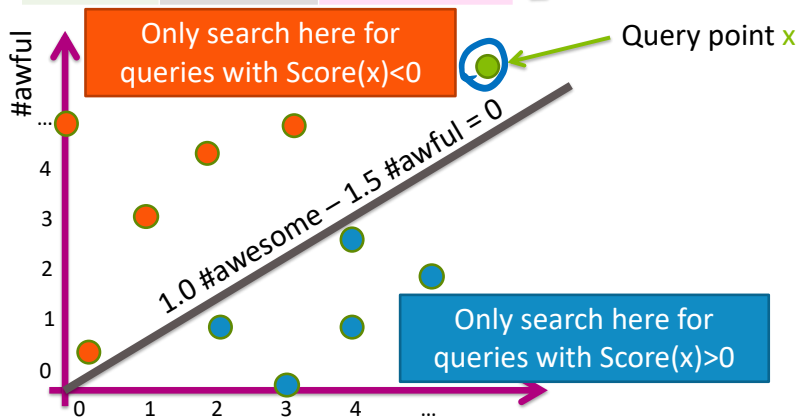
Binning

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 $\text{Score}(x) < 0$



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.

$$O(n/2)$$

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
1	$x_1 = [0, 5]$	-1	0
2	$x_2 = [1, 3]$	-1	0
3	$x_3 = [3, 0]$	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

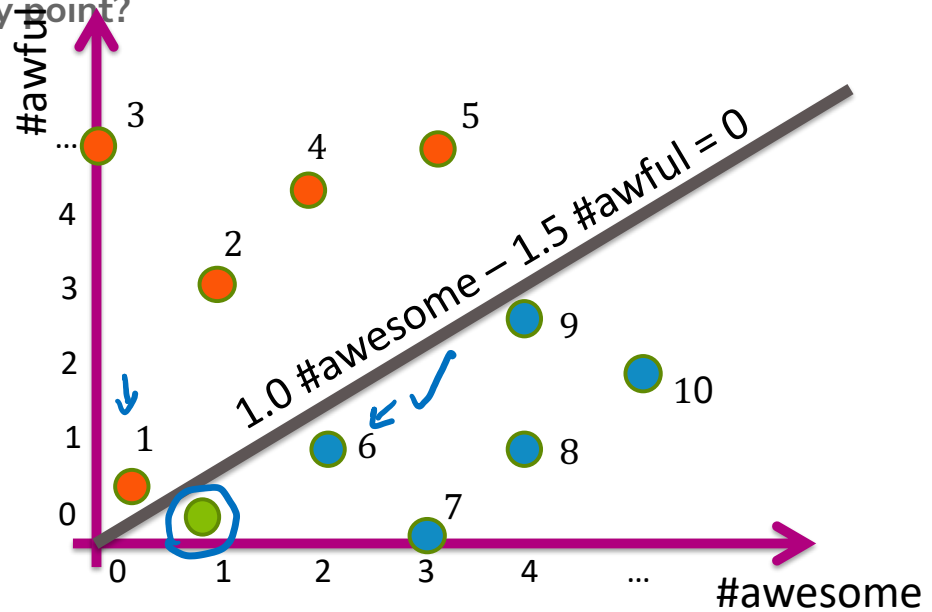
slido

Group 

1 min

slido #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...



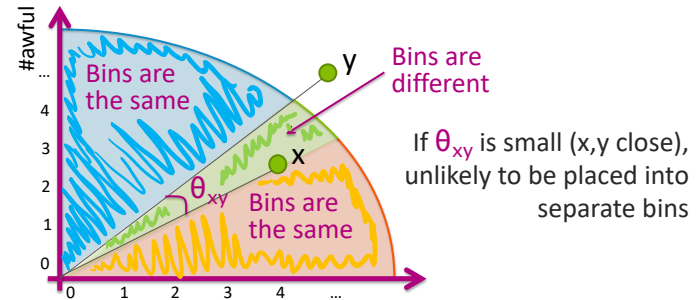
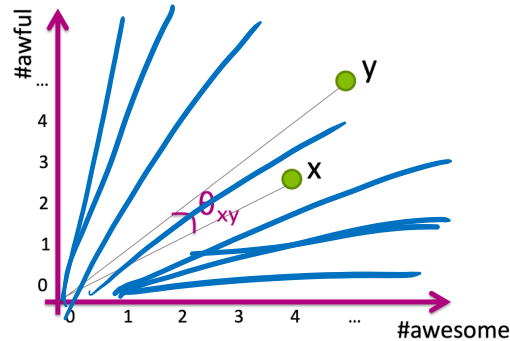
1. How to choose line?

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!



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...



3:30



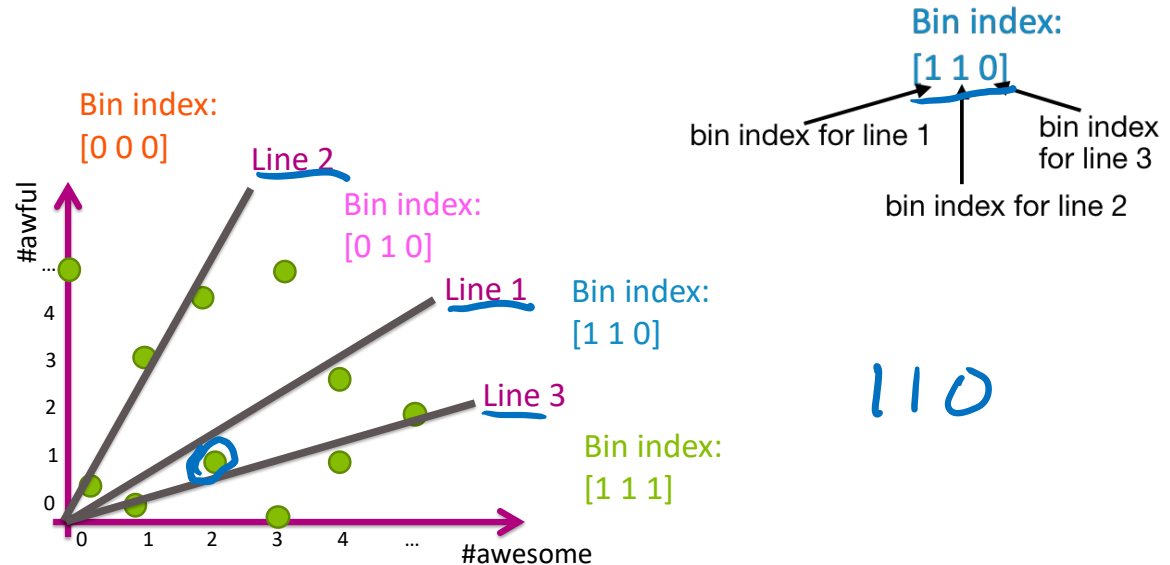
Brain Break



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

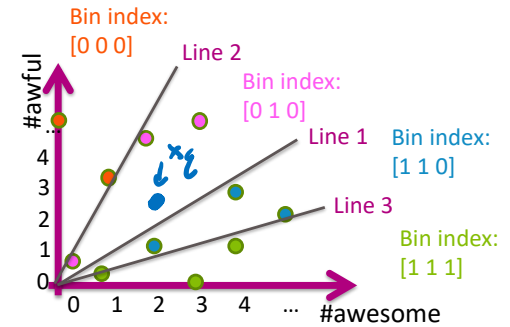
Only search over the points in that bin

LSH Example

$$x_q \rightarrow [0 \ 1 \ 0]$$

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.

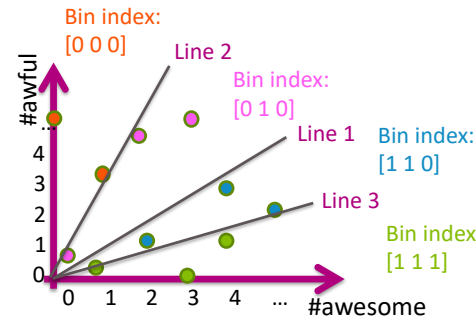
Improve Quality

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

$[0\ 1\ 0\ \dots]$



$\{x_i, x_j, \dots\}$

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.

$[1\ 1\ 0\ \dots]$

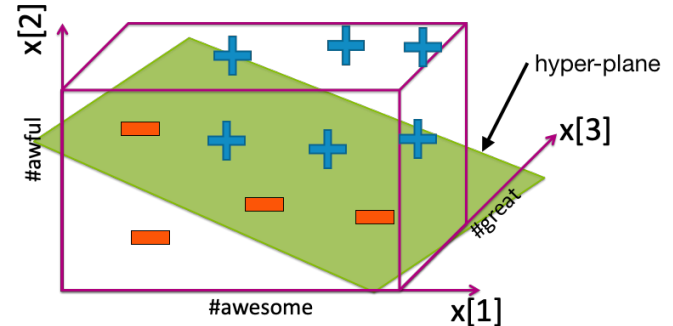
Higher Dimensions

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 $h \approx \log(d)$

$$\text{Score}(x) = w_1 \# \text{awesome} + w_2 \# \text{awful} + w_3 \# \text{great}$$



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



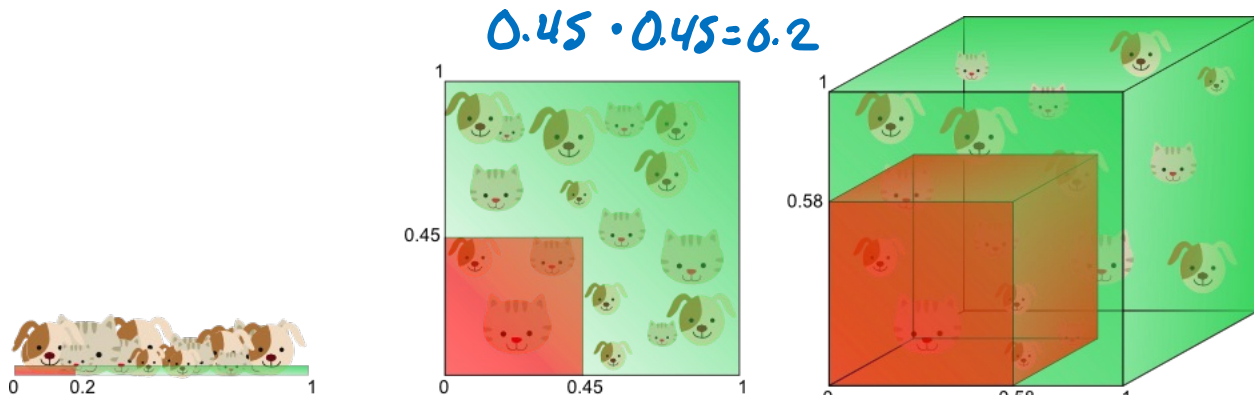
Curse of Dimensionality

High Dimensions

$$2^{100}$$

Methods like k-NN and k-means that rely on computing distances start to struggle in high dimensions.

As the number of dimensions grow, the data gets sparser!



↑
If you have 45%
coverage of 2 Factors, that only covers 20%!

Need more data to make sure you cover all the space in high dim.

$$D = \text{dim}$$

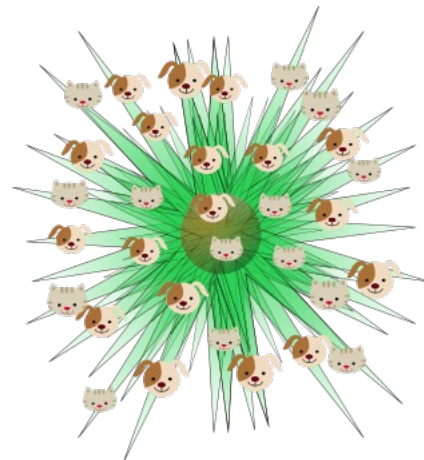
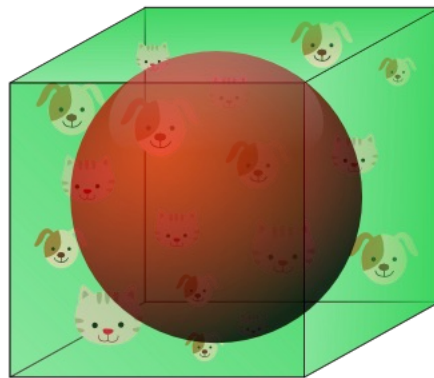
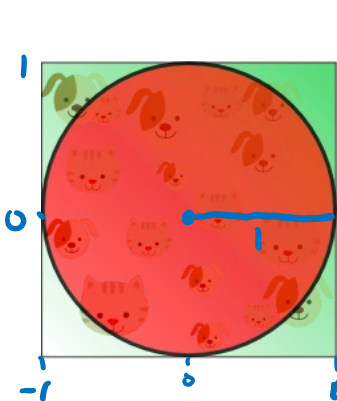
$$n \approx O(2^D)$$

Even Weirder

$$\frac{\pi 1^2}{2^2} \approx \frac{3}{4}$$

$$\frac{\frac{3}{4} 2^3}{2^3} \approx 50\% \rightarrow \text{More mass more than 1 unit dist away}$$

It's believable with more dimensions the data becomes more sparse, but what's even weirder is the sparsity is not uniform!



As D increases, the “mass” of the space goes towards the corners.

Most of the points aren't in the center.

Your nearest neighbors start looking like your farthest neighbors!



Practicalities

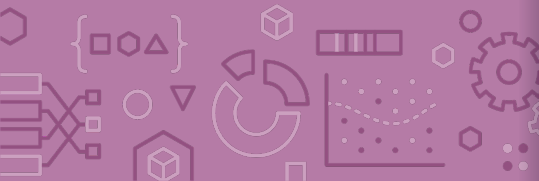
Have you pay attention to the number of dimensions

Very tricky if $n < D$

Can run into some strange results if D is very large

Later, we will talk about ways of trying to do dimensionality reduction in order to reduce the number of dimensions here.

Dimensionality Reduction, PCA
(next week)



HW5 – ML Practice with Kaggle

Kaggle is a website that hosts ML competitions. Very popular among people trying to learn more about ML! Let's you practice a lot of real-world ML skills on challenging problems.

About halfway through the course, want to give you a chance to apply what you've learned in the real-world! HW6 will be hosted as an internal Kaggle competition.

TAs in section tomorrow will walk through how to do submissions, but all instructions are on the Kaggle website (linked from HW spec).

NOT graded as a competition. The leaderboard is just for fun, and we will provide a small amount of EC to the top 3 teams.

Submission and Grading

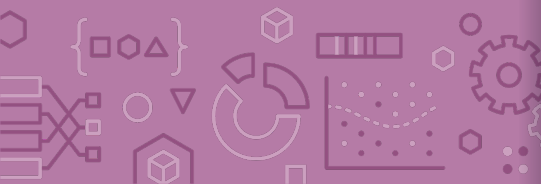
You will need to submit three things:

On Kaggle, submit your predictions for the test set

On Gradescope, submit your Jupyter Notebook that trains/evaluates your models and answers some questions we asked you to answer.

On Gradescope, submit concept questions.

Part of your grade will be from the accuracy on the private test set. Will count 95% accuracy on the private test set as full credit for the performance part of the assignment.



Groups

You will be able to work on the Kaggle portion with a team of up to 3 people. You can work alone if you choose, but we recommend working with a team since that's a great learning experience!

If you work as a team, you can make a team submission on Kaggle and submit your code/report answers on Gradescope together.

Concept portion should be done individually.

