



Recap

Deep Learning  
in Practice

# Pros

No need to manually engineer features, enable automated learning of features

Impressive performance gains

- Image processing

- Natural Language Processing

- Speech recognition

Making huge impacts in most fields



# Cons

Requires a LOT of data

Computationally really expensive

Environmentally, extremely expensive ([Green AI](#))

Hard to tune hyper-parameters

Choice of architecture (we've added even more hyper-parameters)

- Size of kernels, stride, 0 padding, number of conv layers, depth of outputs of conv layers,

Learning algorithm

Still not very interpretable



# NN Failures



**"panda"**  
57.7% confidence

"No one adds noise to things in real applications"

**Not true!**

Hackers will hack

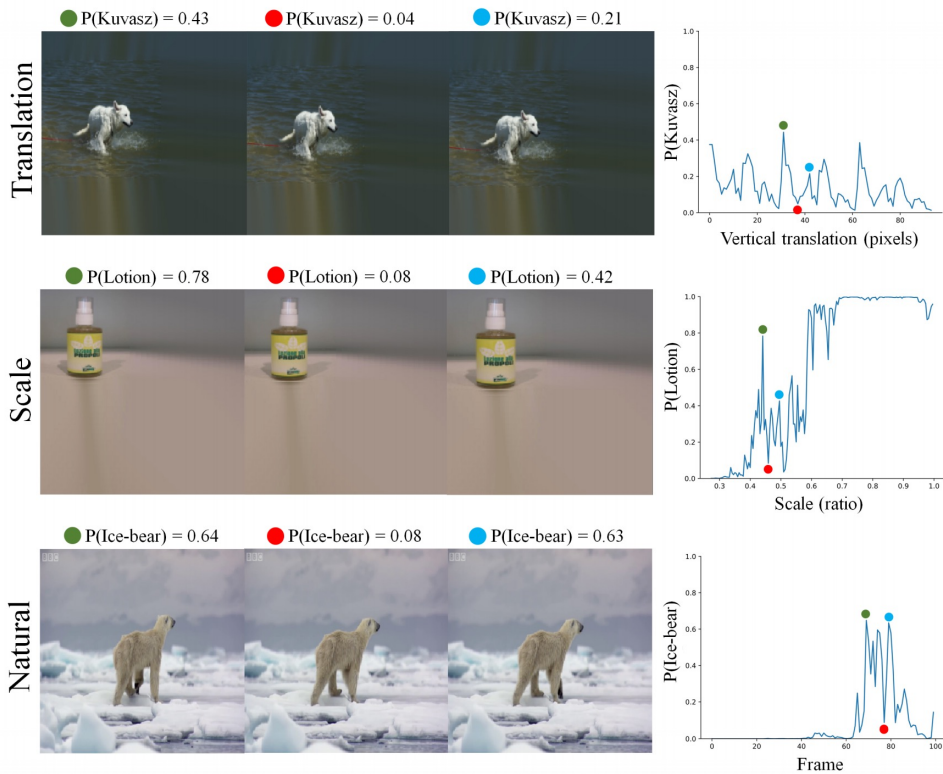
Sensors (cameras) are noisy!



# NN Failures

They even fail with “natural” transformations of images

[Azulay, Weiss <https://arxiv.org/abs/1805.12177>]



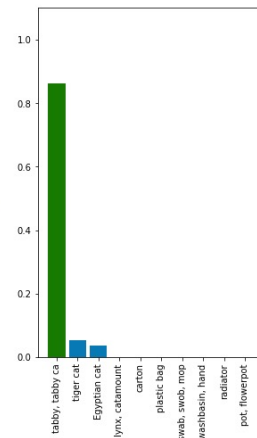
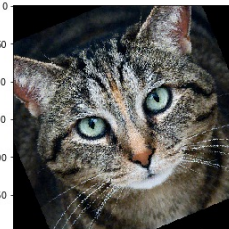
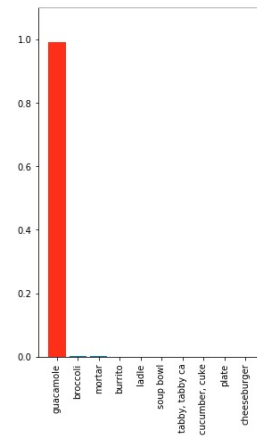
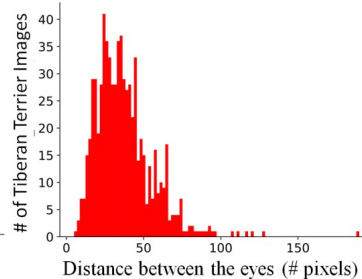
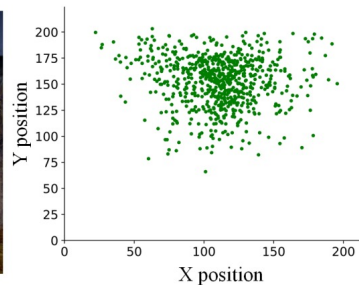
# NN Failures

Objects can be created to trick neural networks!



# Dataset Bias

Datasets, like ImageNet, are generally biased



One approach is to augment your dataset to add random permutations of data to avoid bias.



# Demo: Adversarial Neural Networks to Promote Fairness

<https://godatadriven.com/blog/towards-fairness-in-ml-with-adversarial-networks/>

Dataset: [Adult UCI](#)

- Predict whether a person's income will be  $> \$50K$  or  $\leq \$50K$  based on factors like:
  - Age
  - Education level
  - Marital status
  - Served in Armed Services?
  - Hours per week worked
  - Occupation sector
  - Etc.

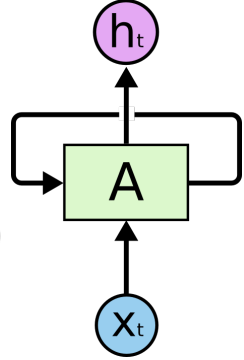
# Further Readings on Deep Learning

Dealing with Variable Length Sequences (e.g. language)

Recurrent Neural Networks (RNNs)

Long Short Term Memory Nets (LSTMs)

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Reinforcement Learning

[Google DeepMind AlphaGo Zero](#)

Generative Adversarial Networks

[How to learn synthetic data](#)

[Green AI](#)

# Pre-Lecture Video

# Detecting Spam

Imagine I made a “Dummy Classifier” for detecting spam

The classifier ignores the input, and always predicts spam.

This actually results in 90% accuracy! Why?

- Most emails are spam...

This is called the **majority class classifier**.

A classifier as simple as the majority class classifier can have a high accuracy if there is a **class imbalance**.

A class imbalance is when one class appears much more frequently than another in the dataset

This might suggest that accuracy isn't enough to tell us if a model is a good model.



# Assessing Accuracy

Always digging in and ask critical questions of your accuracy.

Is there a **class imbalance**?

How does it compare to a baseline approach?

- Random guessing
- Majority class
- ...

Most important: **What does my application need?**

- What's good enough for user experience?
- What is the impact of a mistake we make?



# Confusion Matrix

For binary classification, there are only two types of mistakes

$$\hat{y} = +1, y = -1$$

$$\hat{y} = -1, y = +1$$

Generally we make a **confusion matrix** to understand mistakes.

		Predicted Label	
		+	-
True Label	+	True Positive (TP)	False Negative (FN)
	-	False Positive (FP)	True Negative (TN)

# Binary Classification Measures

Notation

$$C_{TP} = \#TP, \quad C_{FP} = \#FP, \quad C_{TN} = \#TN, \quad C_{FN} = \#FN$$

$$N = C_{TP} + C_{FP} + C_{TN} + C_{FN}$$

$$N_P = C_{TP} + C_{FN}, \quad N_N = C_{FP} + C_{TN}$$

**Error Rate**

$$\frac{C_{FP} + C_{FN}}{N}$$

**Accuracy Rate**

$$\frac{C_{TP} + C_{TN}}{N}$$

**False Positive rate (FPR)**

$$\frac{C_{FP}}{N_N}$$

**False Negative Rate (FNR)**

$$\frac{C_{FN}}{N_P}$$

**True Positive Rate or  
Recall**

$$\frac{T_P}{N_P}$$

**Precision**

$$\frac{T_P}{C_{TP} + C_{FP}}$$

**F1-Score**

$$2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

[See more!](#)

# Change Threshold

What if I never want to make a false positive prediction?

What if I never want to make a false negative prediction?

One way to control for our application is to change the scoring threshold. (Could also change intercept!)

If  $Score(x) > \alpha$ :

- Predict  $\hat{y} = +1$

Else:

- Predict  $\hat{y} = -1$





# ROC Curve

What happens to our TPR and FPR as we increase the threshold?



# Assessing Accuracy

Often with binary classification, we treat the positive label as being the more important of the two. We then often then focus on these metrics:

**Precision:** Of the ones I predicted positive, how many of them were actually positive?

**Recall:** Of all the things that are truly positive, how many of them did I correctly predict as positive?



# Precision

What fraction of the examples I predicted positive were correct?

Sentences predicted to be positive:

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

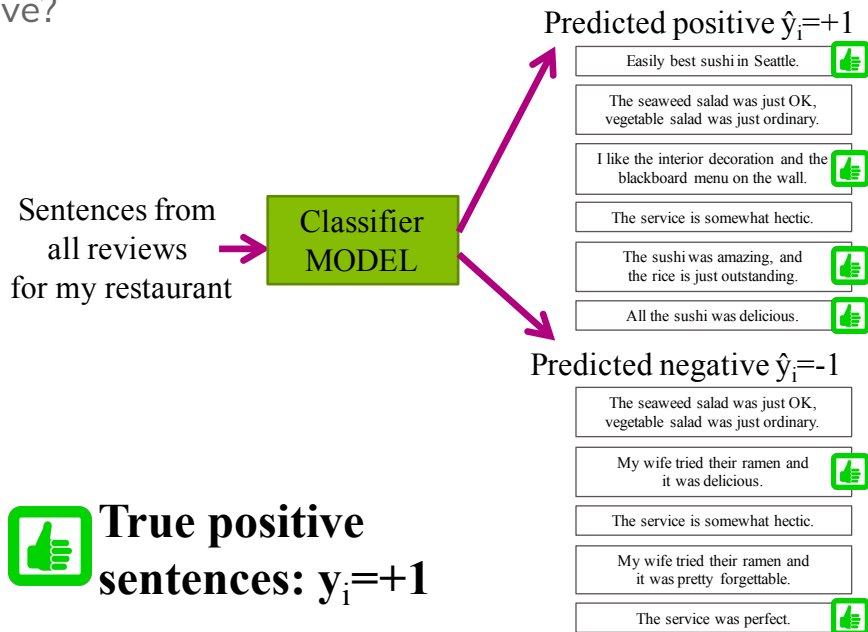
Easily best sushi in Seattle.	✓
The seaweed salad was just OK, vegetable salad was just ordinary.	✗
I like the interior decoration and the blackboard menu on the wall.	✓
The service is somewhat hectic.	✗
The sushi was amazing, and the rice is just outstanding.	✓
All the sushi was delicious.	✓

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

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

# Recall

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



$$recall = \frac{C_{TP}}{N} = \frac{C_{TP}}{C_{TP} + C_{FP}}$$

# Precision & Recall

An optimistic model will predict almost everything as positive

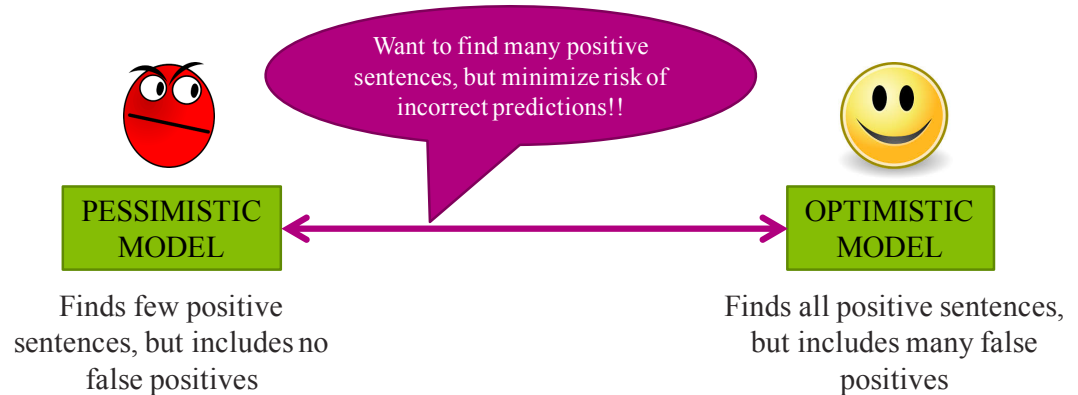


High recall, low precision

A pessimistic model will predict almost everything as negative



High precision, low recall



# Controlling Precision/Recall

Depending on your application, precision or recall might be more important

Ideally you will have high values for both, but generally increasing recall will decrease precision and vice versa.

For logistic regression, we can control for how optimistic the model is by changing the threshold for positive classification

**Before**

$$\hat{y}_i = +1 \text{ if } \hat{P}(y = +1|x_i) > 0.5 \text{ else } \hat{y}_i = -1$$

**Now**

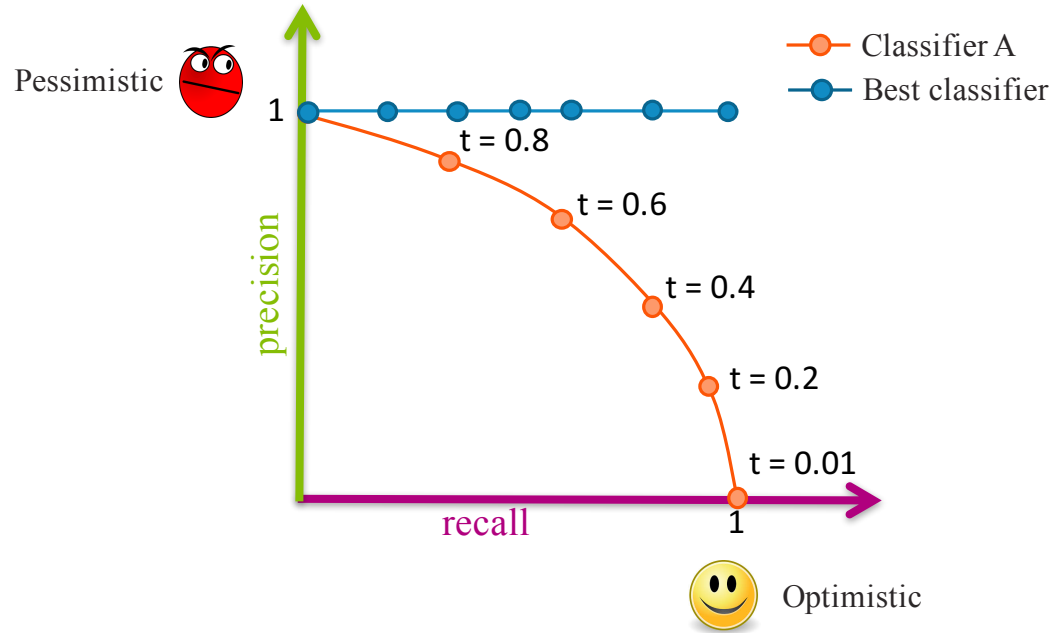
$$\hat{y}_i = +1 \text{ if } \hat{P}(y = +1|x_i) > t \text{ else } \hat{y}_i = -1$$

# Precision- Recall Curve



# Precision-Recall Curve

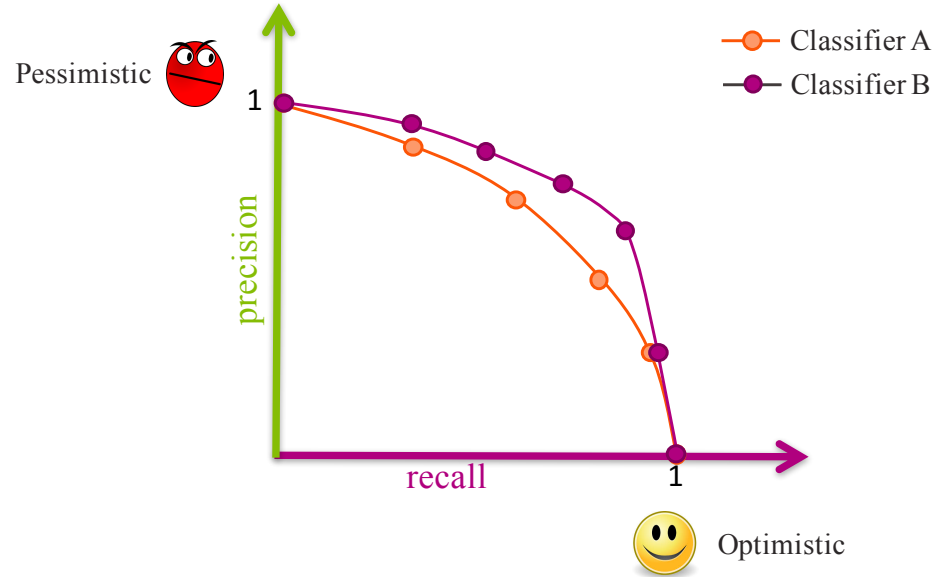
Can try every threshold to get a curve like below





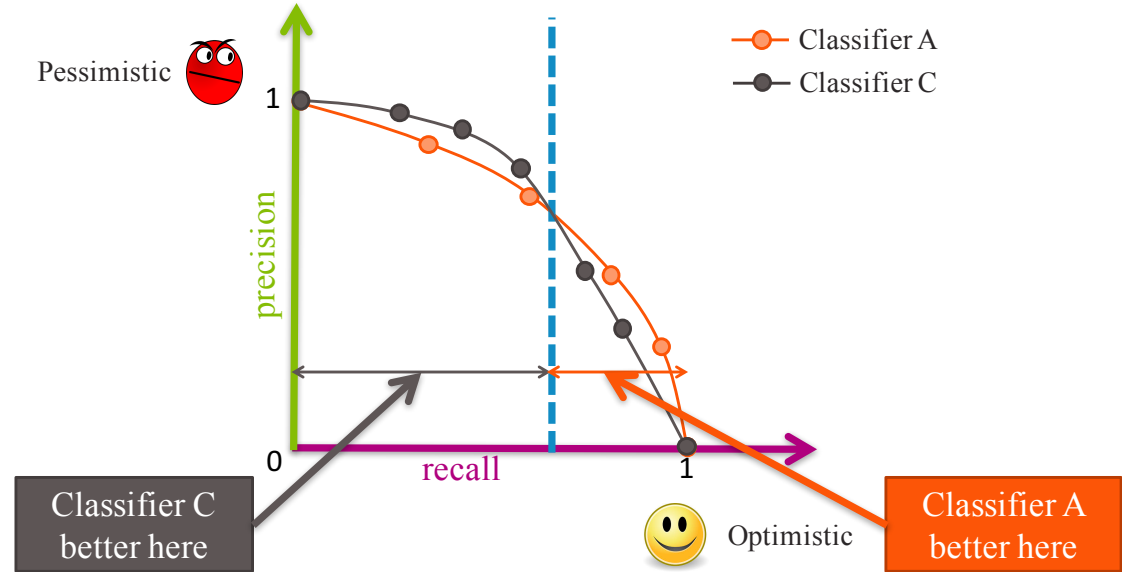
# Precision-Recall Curve

Sometimes, Classifier B is strictly better than Classifier A



# Precision-Recall Curve

Most times, the classifiers are incomparable



# Compare Classifiers

Often come up with a single number to describe it

F1-score, AUC, etc.

Remember, what your application needs is most important





Also common to use **precision at k**

If you show the top **k** most likely positive examples, how many of them are true positives

Showing  
k=5 sentences  
on website



Sentences model  
most sure are positive

Easily best sushi in Seattle.	
My wife tried their ramen and it was pretty forgettable.	
The sushi was amazing, and the rice is just outstanding.	
All the sushi was delicious.	
The service was perfect.	



precision at k = 0.8



# Roadmap

1. Housing Prices - Regression
  - Regression Model
  - Assessing Performance
  - Ridge Regression
  - LASSO
2. Sentiment Analysis – Classification
  - Classification Overview
  - Logistic Regression
  - Bias / Fairness
  - Decision Trees
  - Ensemble Methods
3. Deep Learning
  - Neural Networks
  - Convolutional Neural
4. Document Retrieval – Clustering and Similarity
  - Precision / Recall
  - k-Nearest Neighbor
  - Kernel Methods
  - Locality Sensitive Hashing
  - Clustering
  - Hierarchical Clustering



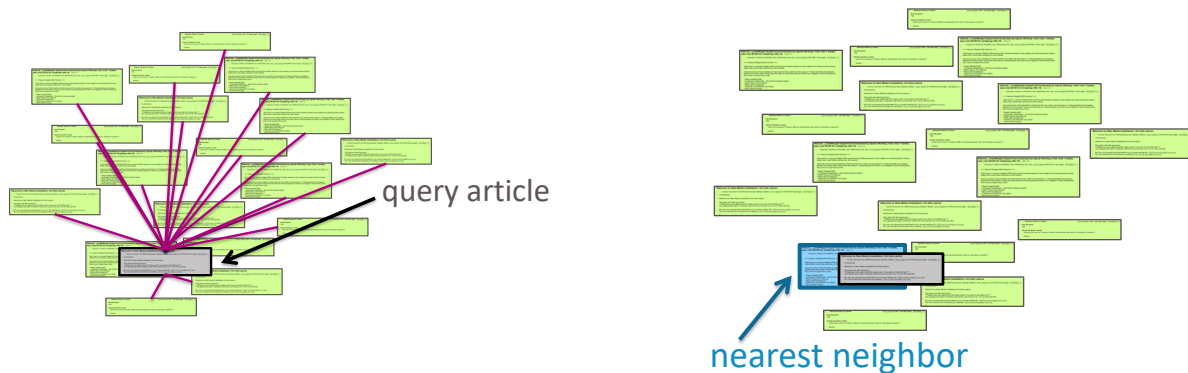
# Document Retrieval

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?

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



# Nearest Neighbors

# 1-Nearest Neighbor

## Input

$x_q$ : Query example (e.g. my book)

$x_1, \dots, x_n$ : Corpus of documents (e.g. Amazon books)

## Output

The document in corpus that is most similar to  $x_q$

$$x^{NN} = \arg \min_{x_i \in [x_1, \dots, x_n]} \text{distance}(x_q, x_i)$$

It's very critical to properly define how we represent each document  $x_i$  and the similarity metric *distance*! Different definitions will lead to very different results.



# 1-Nearest Neighbor

How long does it take to find the 1-NN? About  $n$  operations

**Input:**  $x_q$

$x^{NN} = \emptyset$

$nn\_dist = \infty$

for  $x_i \in [x_1, \dots, x_n]$ :

$dist = distance(x_q, x_i)$

if  $dist < nn\_dist$ :

$x^{NN} = x_i$

$nn\_dist = dist$

**Output:**  $x^{NN}$

# k-Nearest Neighbors

## Input

$x_q$ : Query example (e.g. my book)

$x_1, \dots, x_n$ : Corpus of documents (e.g. Amazon books)

## Output

List of  $k$  documents most similar to  $x_q$

Formally



# k-Nearest Neighbors

Same idea as 1-NN algorithm, but maintain list of k-NN

***Input:***  $x_q$

$X^{kNN} = [x_1, \dots, x_k]$

$nn\_dists = [dist(x_1, x_q), dist(x_2, x_q), \dots, dist(x_k, x_q)]$

for  $x_i \in [x_{k+1}, \dots, x_n]$ :

$dist = distance(x_q, x_i)$

    if  $dist < \max(nn\_dists)$ :

        remove largest dist from  $X^{kNN}$  and  $nn\_dists$

        add  $x_i$  to  $X^{kNN}$  and  $distance(x_q, x_i)$  to  $nn\_dists$

***Output:***  $X^{kNN}$

# k-Nearest Neighbors

Can be used in many circumstances!

## Retrieval

Return  $X^{k-NN}$

## Regression

$$\hat{y}_i = \frac{1}{k} \sum_{j=1}^k x^{NN_j}$$

## Classification

$$\hat{y}_i = \text{majority\_class}(X^{k-NN})$$



# slido

Group 

1.5 min

[sli.do #cs416](https://sli.do/#cs416)

In the regression/classification settings, what is the relationship between  $k$  for  $k$ -NN and the bias/variance of the model? Each option completes the sentence “As  $k$  increases ...”

Bias increases, Variance increases

Bias decreases, Variance increases

Bias increases, Variance decreases

Bias decreases, Variance decreases



## *Brain Break*



# Embeddings

# Important Points

While the formalization of these algorithms can be a bit tedious, the intuition is fairly simple. Find the 1 or  $k$  nearest neighbors to a given document and return those as the answer.

This intuition relies on answering two important questions

How do we represent the documents  $x_i$ ?

How do we measure the distance  $distance(x_q, x_i)$ ?





# Document Representation

Like our previous ML algorithms, we will want to make a vector out of the document to represent it as a point in space.

Simplest representation is the **bag-of-words** representation.

Each document will become a  $W$  dimension vector where  $W$  is the number of words in the entire corpus of documents

The value of  $x_i[j]$  will be the number of times word  $j$  appears in document  $i$ .

This ignores order of words in the document, just the counts.



# Bag of Words

## Pros

Very simple to describe

Very simple to compute

## Cons

Common words like “the” and “a” dominate counts of uncommon words

Often it’s the uncommon words that uniquely define a doc.

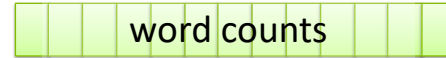


# TF-IDF

**Goal:** Emphasize important words

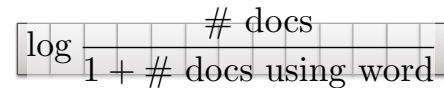
Appear frequently in the document (common locally)

Term frequency =



Appears rarely in the corpus (rare globally)

Inverse doc freq. =



tf \* idf

Do a pair-wise multiplication to compute the TF-IDF for each word

Words that appear in every document will have a small IDF making the TF-IDF small!

# slido

Group 

1.5 min

What is the  $TF - IDF("rain", Doc_1)$  with the following documents (assume standard pre-processing)

Doc 1: It is going to rain today.

Doc 2: Today I am not going outside.

Doc 3: I am going to watch the season premiere.

Distance

# Euclidian Distance

Now we will define what similarity/distance means

Want to define how “close” two vectors are. A smaller value for distance means they are closer, a large value for distance means they are farther away.

The simplest way to define distance between vectors is the **Euclidean distance**

$$\begin{aligned} \text{distance}(x_i, x_q) &= \|x_i - x_q\|_2 \\ &= \sqrt{\sum_{j=1}^D (x_i[j] - x_q[j])^2} \end{aligned}$$

# Manhattan Distance

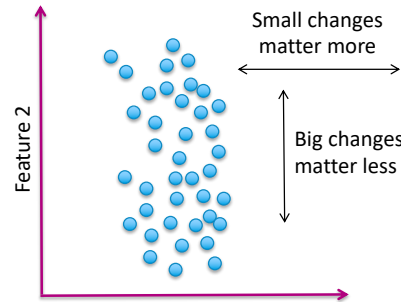
Another common choice of distance is the **Manhattan Distance**

$$\begin{aligned} \text{distance}(x_i, x_q) &= \left\| x_i - x_q \right\|_1 \\ &= \sum_{j=1}^D |x_i[j] - x_q[j]| \end{aligned}$$



# Weighted Distances

Some features vary more than others or are measured in different units. We can weight different dimensions differently to make the distance metric more reasonable.



Specify weights as  
a function of  
feature spread

For feature  $j$ : 
$$\frac{1}{\max_i(x_i[j]) - \min_i(x_i[j])} = a_j$$

Weighted Euclidean distance

$$distance(x_i, x_q) = \sqrt{\sum_{j=1}^D a_j^2 (x_i[j] - x_q[j])^2}$$



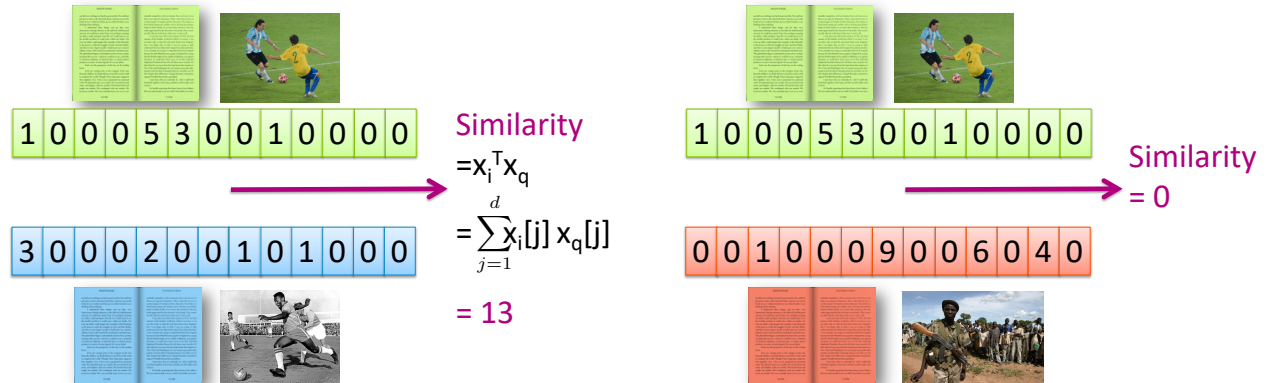
# Similarity

Another natural similarity measure would use

$$x_i^T x_q = \sum_{j=1}^D x_i[j] x_q[j]$$

Notice this is a measure of similarity, not distance

This means a bigger number is better



# Cosine Similarity

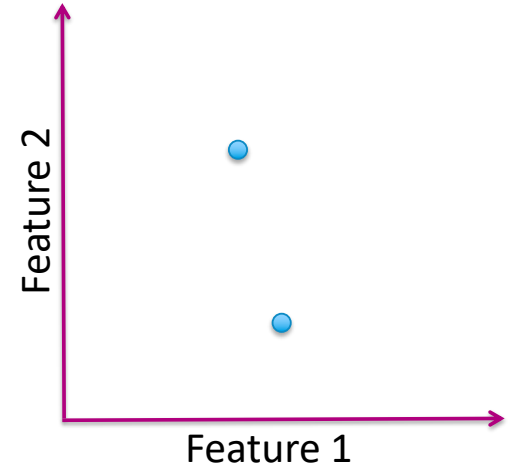
Should we normalize the vectors before finding the similarity?

$$\text{similarity} = \frac{x_i^T x_q}{\|x_i\|_2 \|x_q\|_2} = \cos(\theta)$$

Note:

Not a true distance metric

Efficient for sparse vectors!



# Cosine Similarity

In general

$$-1 \leq \textit{cosine similarity} \leq 1$$

For positive features (like TF-IDF)

$$0 \leq \textit{cosine similarity} \leq 1$$

Define

$$\textit{distance} = 1 - \textit{similarity}$$



# To Normalize or Not To Normalize?

Not normalized



1 0 0 0 5 3 0 0 1 0 0 0 0

Similarity = 13

3 1 0 0 2 0 0 1 0 1 0 0 0



2 0 0 0 10 6 0 0 2 0 0 0 0

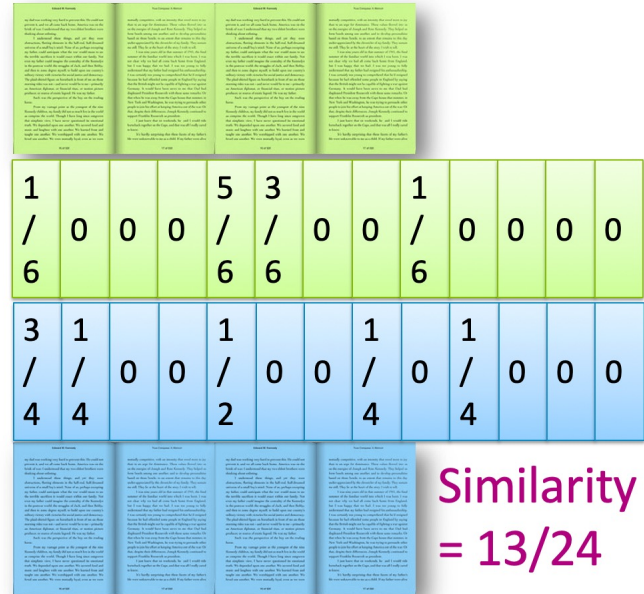
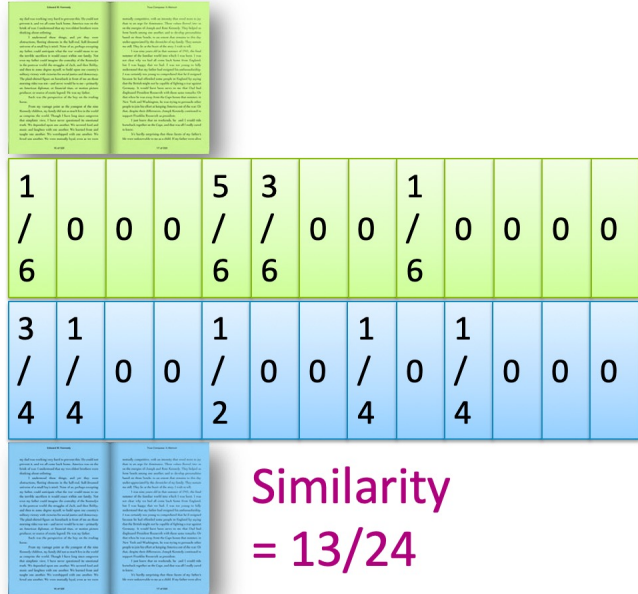
Similarity = 52

6 2 0 0 4 0 0 2 0 2 0 0 0



# To Normalize or Not To Normalize?

## Normalized



## To Normalize or Not To Normalize?

Normalization is not desired when comparing documents of different sizes since it ignores length.



long document



short tweet

**Normalizing can  
make dissimilar  
objects appear more  
similar**



long document



long document

**Common  
compromise:  
Just cap maximum  
word counts**

In practice, can use multiple distance metrics and combine them using some defined weights

# slido

Group 

3 min

[sli.do #cs416](https://sli.do/#cs416)

Not a real Poll Everywhere question, just time to work!

**For the given documents, what are their Euclidean Distance and Cosine Similarity?**

Assume we are using a bag of words representation

Document 1: “I really like dogs”

Document 2: “dogs are really really awesome”

Steps:

Write out bag of words vectors

Compute Euclidean distance

Compute Cosine similarity

# slido

Think 

[sli.do](#) #cs416



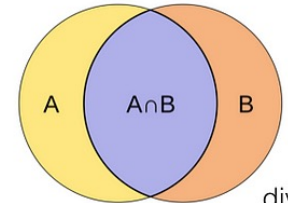
# Jaccard Similarity

Yet another popular similarity measure for text documents.  
Compare the overlap of words appearing in both documents

$$J(Doc_i, Doc_j) = \frac{|Doc_i \cap Doc_j|}{|Doc_i \cup Doc_j|}$$

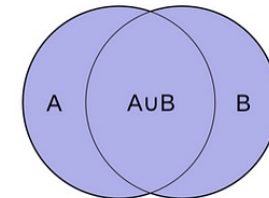
Where  $Doc_i$  and  $Doc_j$  are sets of words in each doc

The intersect of A & B



division

The union of A & B



# Recap

**Theme:** Use nearest neighbors to recommend documents.

**Ideas:**

Precision and Recall Curves

Implement a nearest neighbor algorithm

Compare and contrast different document representations

- Emphasize important words with TF-IDF

Compare and contrast different measurements of similarity

- Euclidean and weighted Euclidean
- Cosine similarity and inner-product similarity





# Precision

What fraction of the examples I predicted positive were correct?

Sentences predicted to be positive:

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

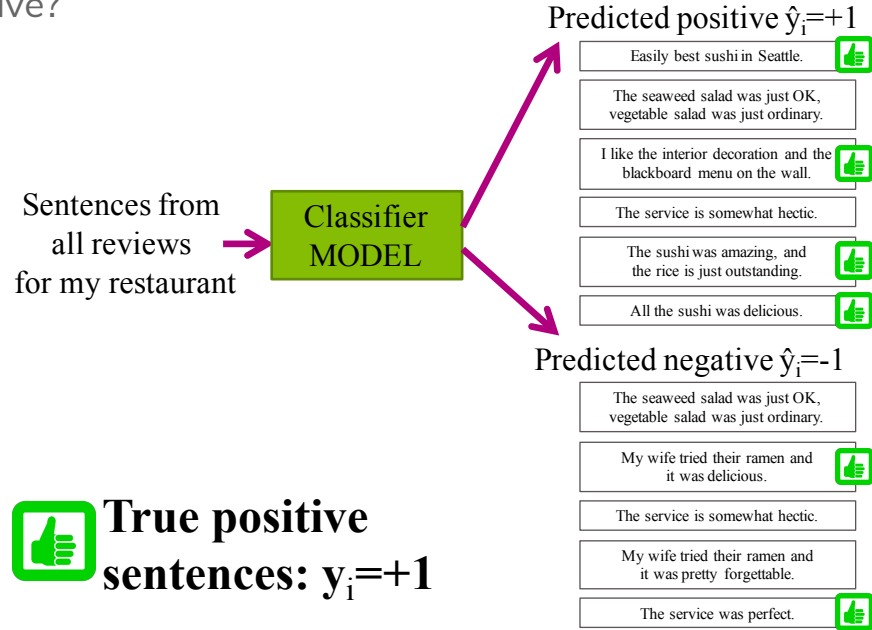
Easily best sushi in Seattle.	✓
The seaweed salad was just OK, vegetable salad was just ordinary.	✗
I like the interior decoration and the blackboard menu on the wall.	✓
The service is somewhat hectic.	✗
The sushi was amazing, and the rice is just outstanding.	✓
All the sushi was delicious.	✓

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

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

# Recall

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



$$recall = \frac{C_{TP}}{N} = \frac{C_{TP}}{C_{TP} + C_{FP}}$$

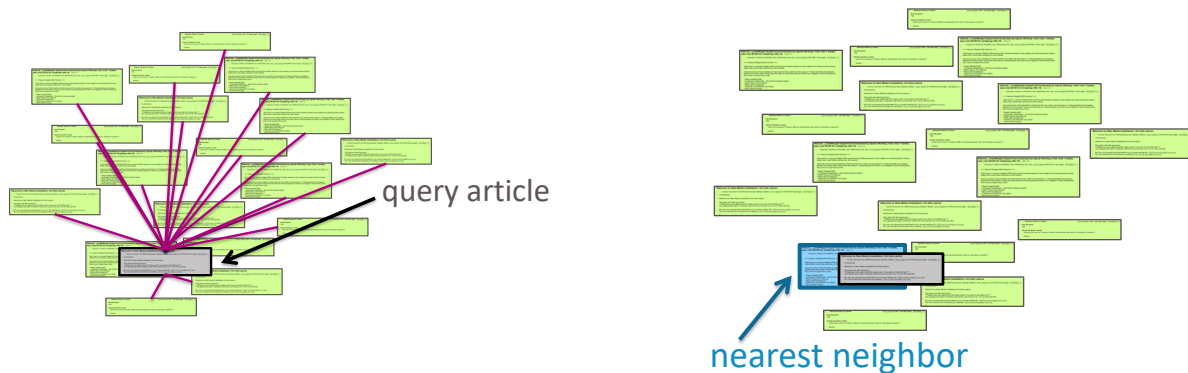
# Document Retrieval

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

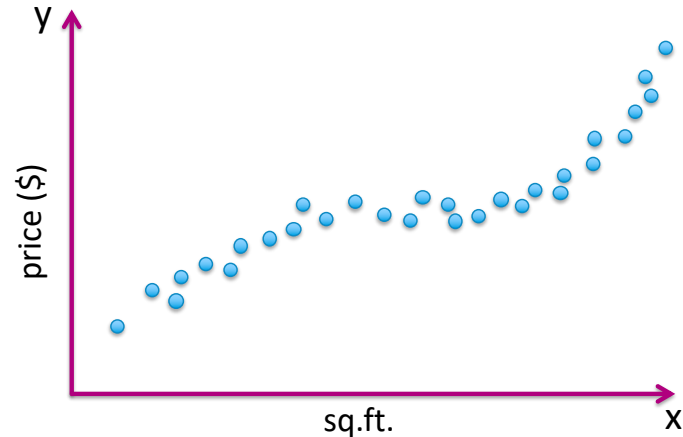
- How do we measure similarity?
- How do we search over books?
- How do we measure accuracy?

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



# Predicting House Prices

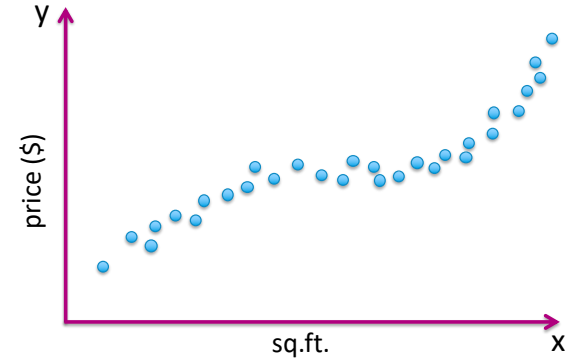
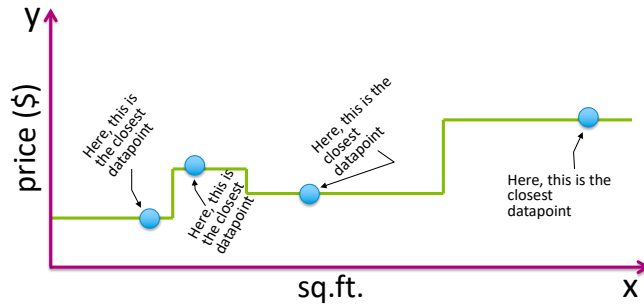
When we saw regression before, we assumed there was some linear/polynomial function that produced the data. All we had to do was choose the right polynomial degree.



# 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:  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

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

**Output:**  $y^{NN}$

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



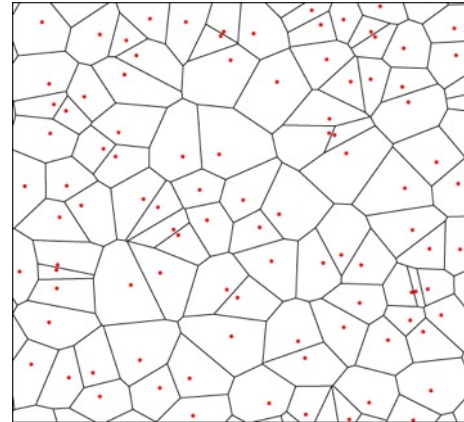
# Visualizing 1-NN Regression

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

Can visualize this with a Voronoi Tessellation

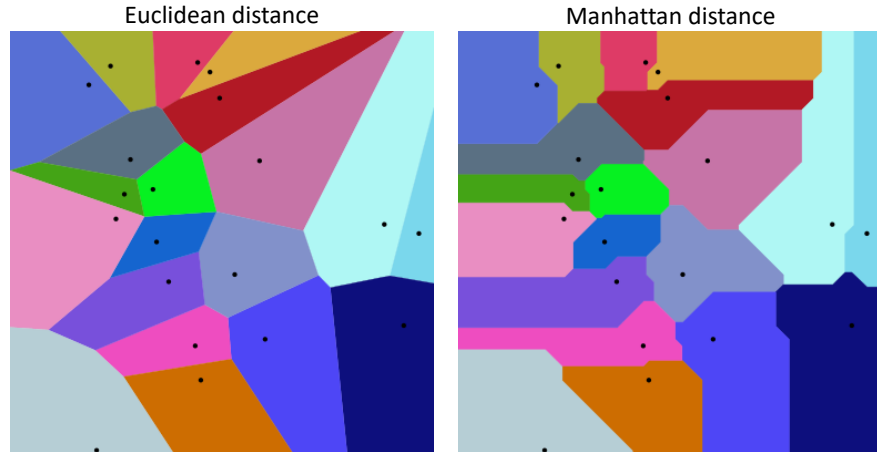
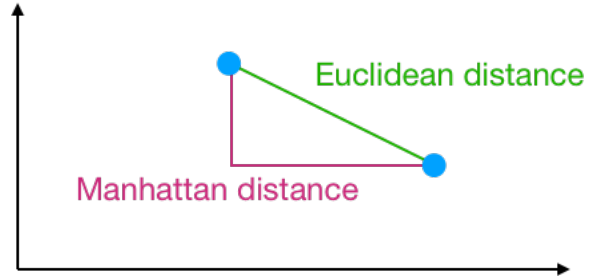
Shows all of the points that are “closest” to a particular training point

Not actually computed in practice, but helps understand



# Visualizing 1-NN Regression

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

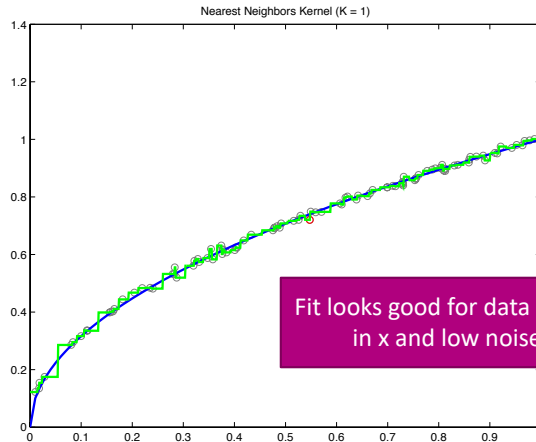


# 1-NN Regression

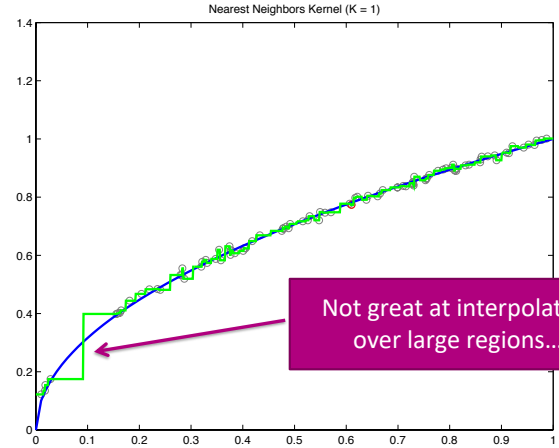
## Weaknesses

Inaccurate if data is sparse

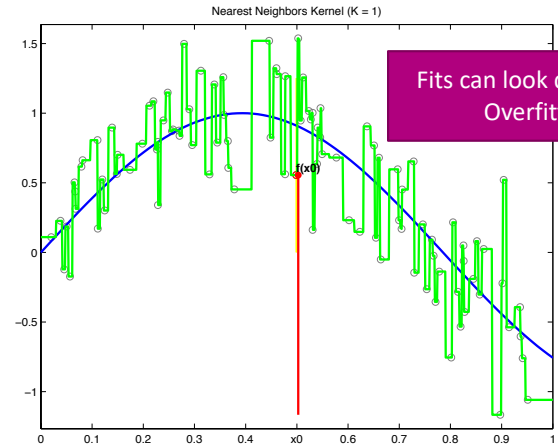
Can wildly overfit



Fit looks good for data dense  
in x and low noise



Not great at interpolating  
over large regions...

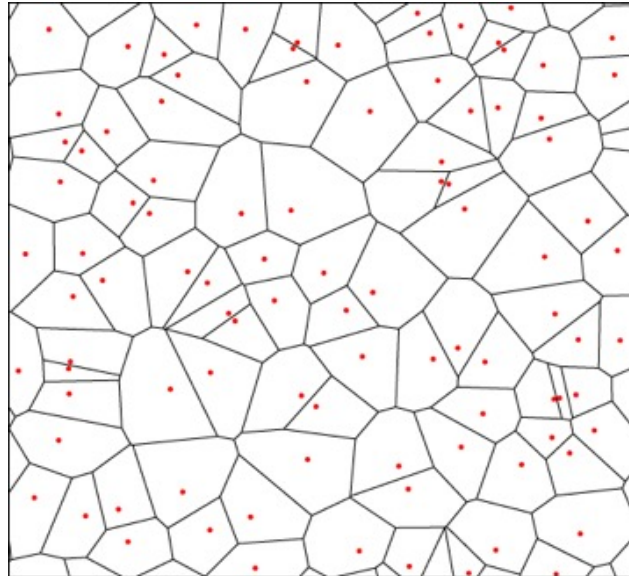


Fits can look quite wild...  
Overfitting?

# 1-NN Classification

Can we use the same algorithm for classification? Yes!

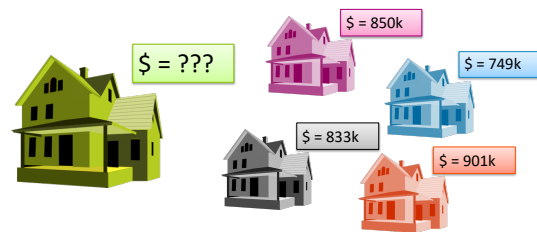
Predict the class of the nearest neighbor. Besides that, exactly the same as regression.



# 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!

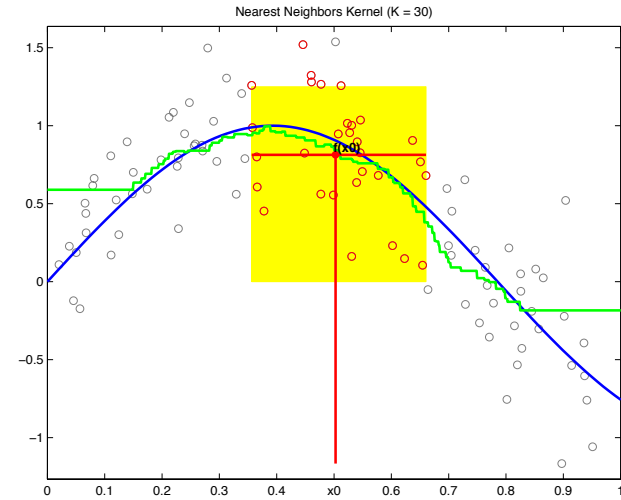
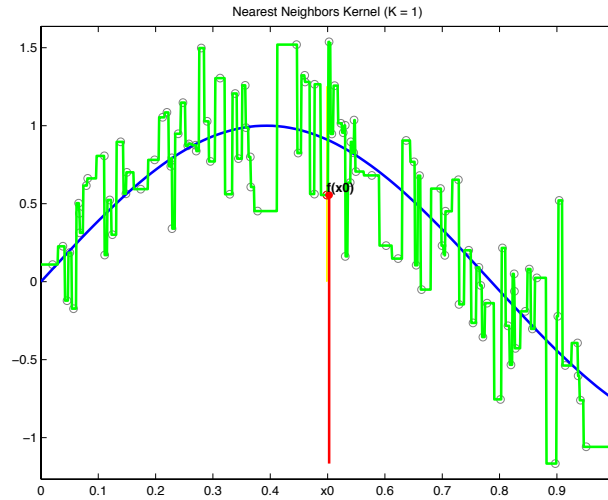


**Input:** Query point:  $x_q$ , Training Data:  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

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

**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

**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  $c_{q,NN_j}$ ?

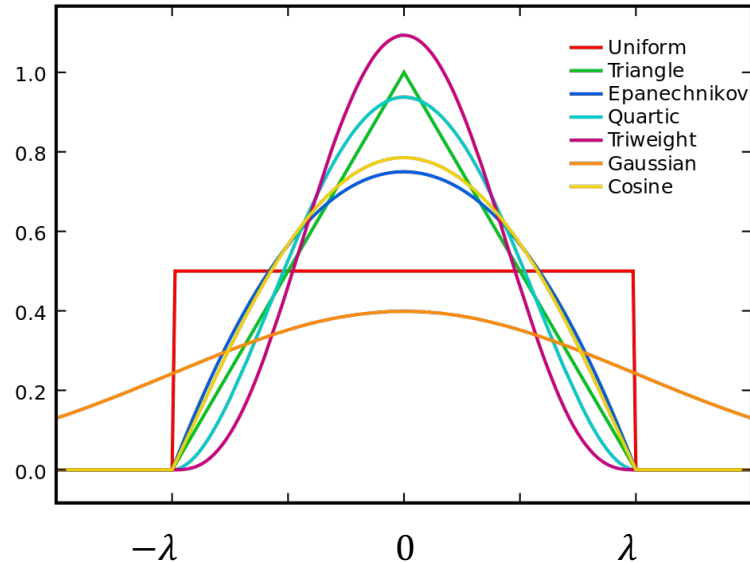
Want  $c_{q,NN_j}$  to be **small** if  
 $dist(x_q, x^{NN_j})$  is **large**.

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

# Kernels

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

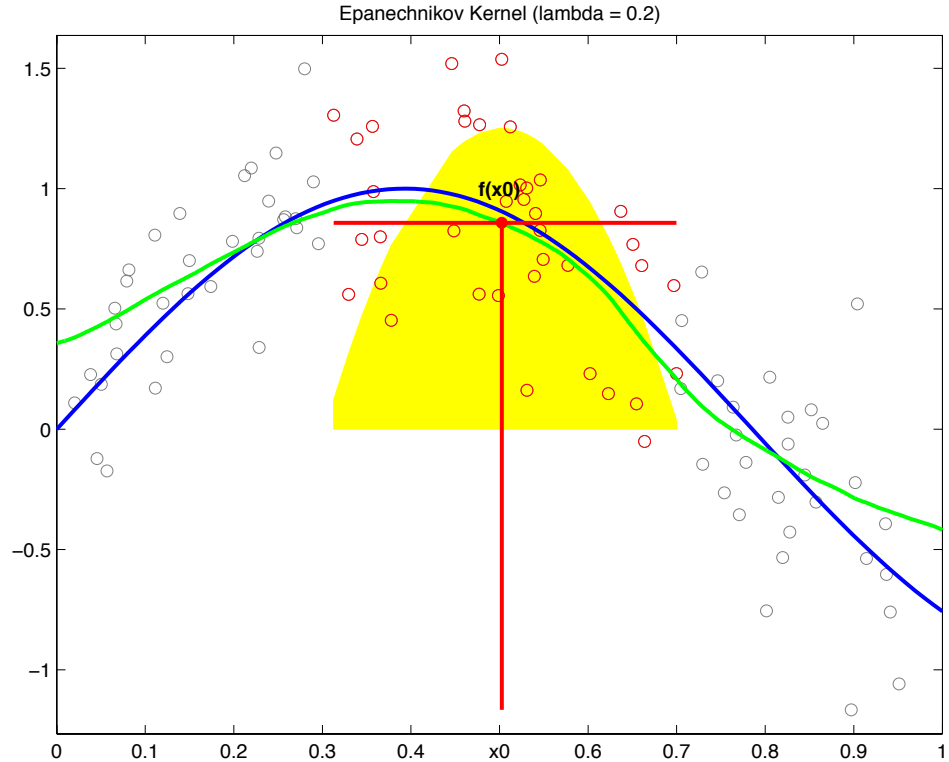
We can take this one step farther, instead of just using a kernel to weight the  $k$  nearest neighbors, can use the kernel to weight **all training points!** This is called **kernel regression**.

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



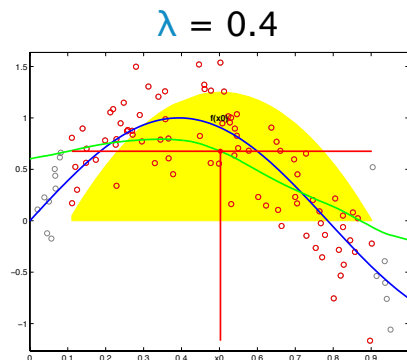
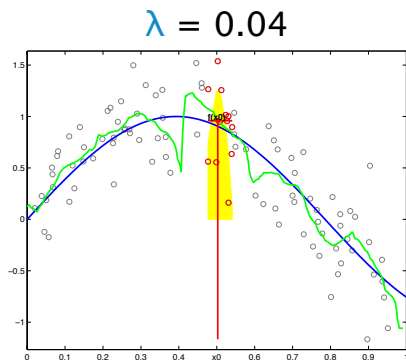
# Visualizing Kernel Regression

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



# Choose Bandwidth $\lambda$

Often, which kernel you use matters much less than which value you use for the bandwidth  $\lambda$



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

Kernel

Bandwidth

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

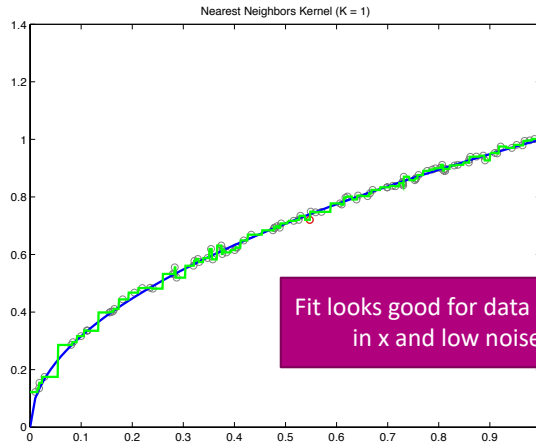


# 1-NN Regression

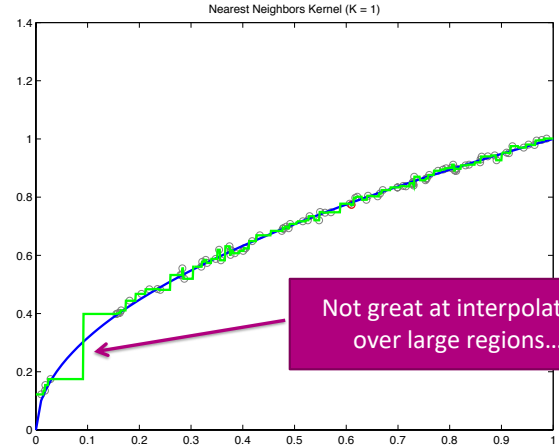
## Weaknesses

Inaccurate if data is sparse

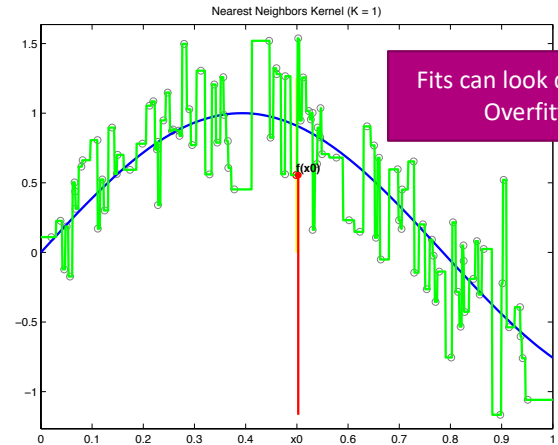
Can wildly overfit



Fit looks good for data dense  
in x and low noise



Not great at interpolating  
over large regions...



Fits can look quite wild...  
Overfitting?

# Weighted k-NN

**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**.

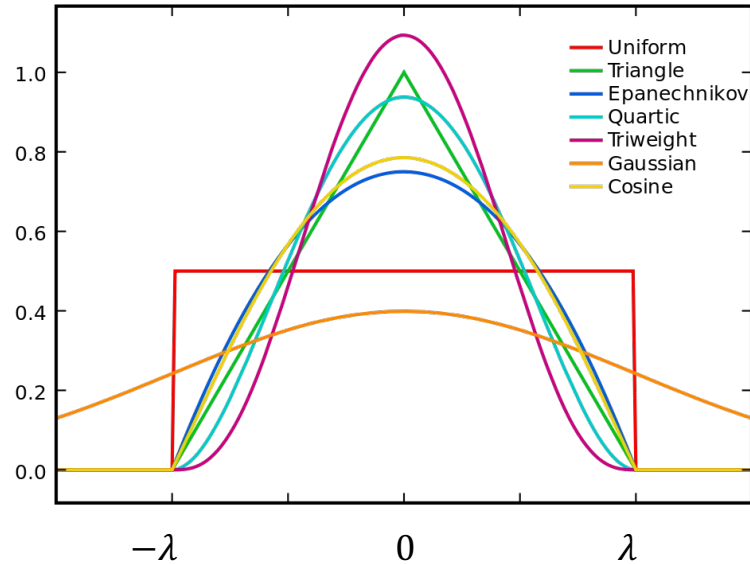
Want  $c_{q,NN_j}$  to be **large** if  
 $dist(x_q, x^{NN_j})$  is **small**.



# Kernels

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

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

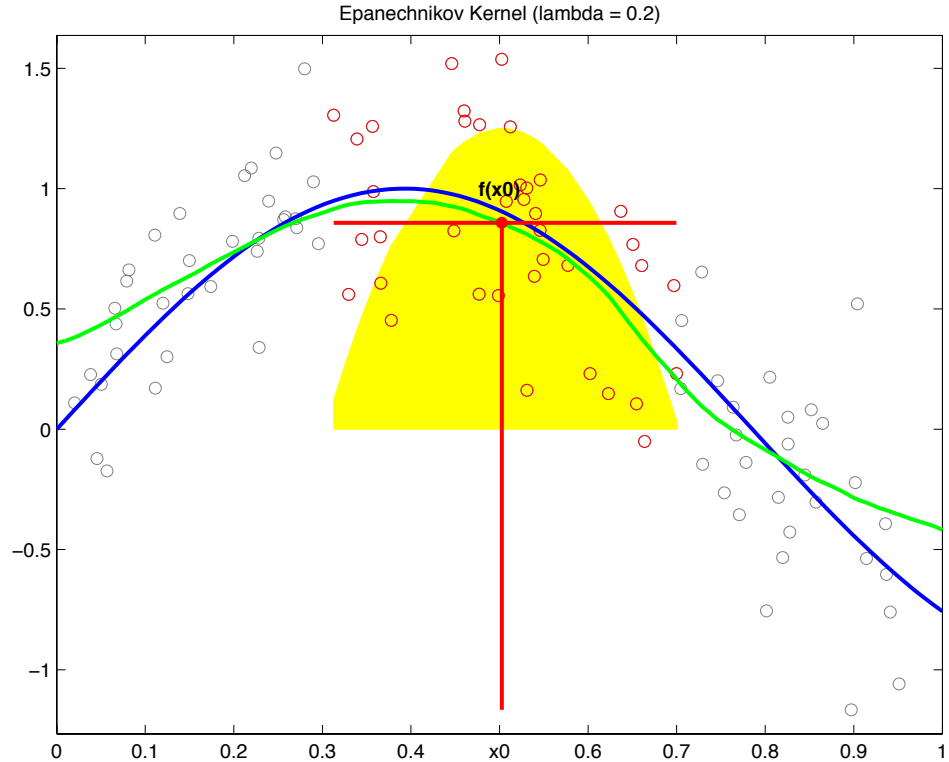


Gaussian Kernel

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

# Visualizing Kernel Regression

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



# Poll Everywhere

Think 

1.5 min

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

k-Nearest Neighbor Regression

Weighted k-Nearest Neighbor Regression

Kernel Regression

[pollev.com/cs416](https://pollev.com/cs416)

# Poll Everywhere

Think 

3 min

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

k-Nearest Neighbor Regression

Weighted k-Nearest Neighbor Regression

Kernel Regression

[pollev.com/cs416](https://pollev.com/cs416)

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



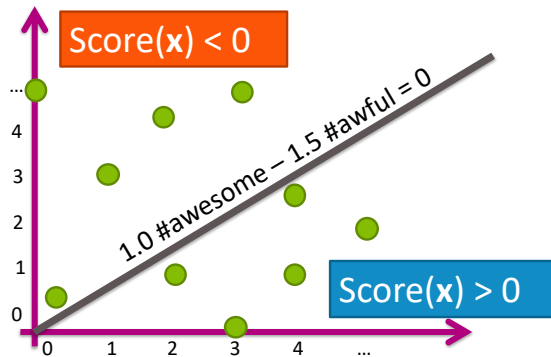


# Binning

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.

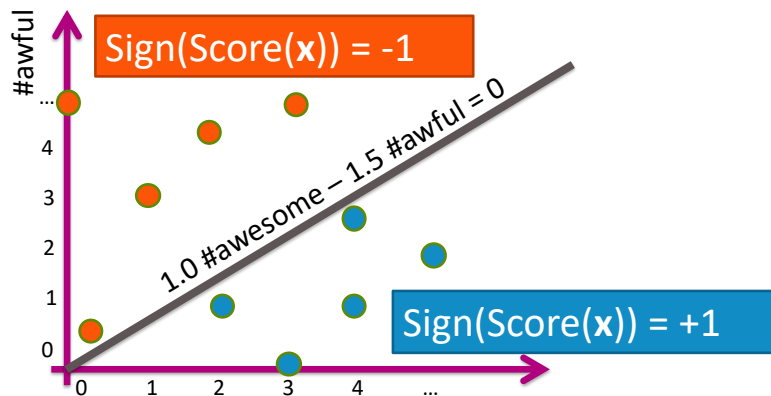


# Binning

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
...	...



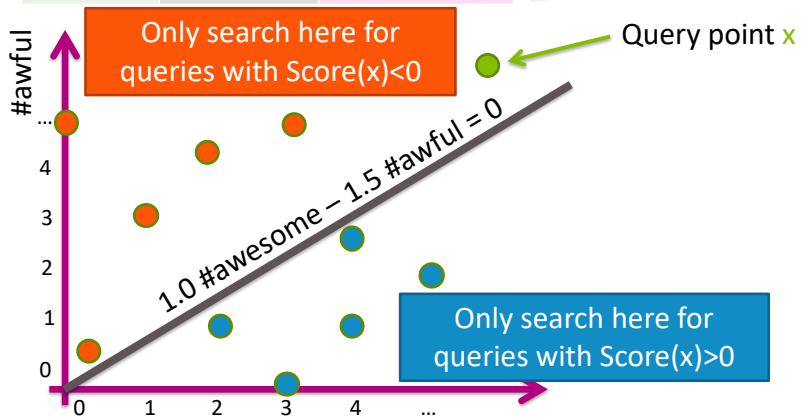
# Binning

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



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.

# 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

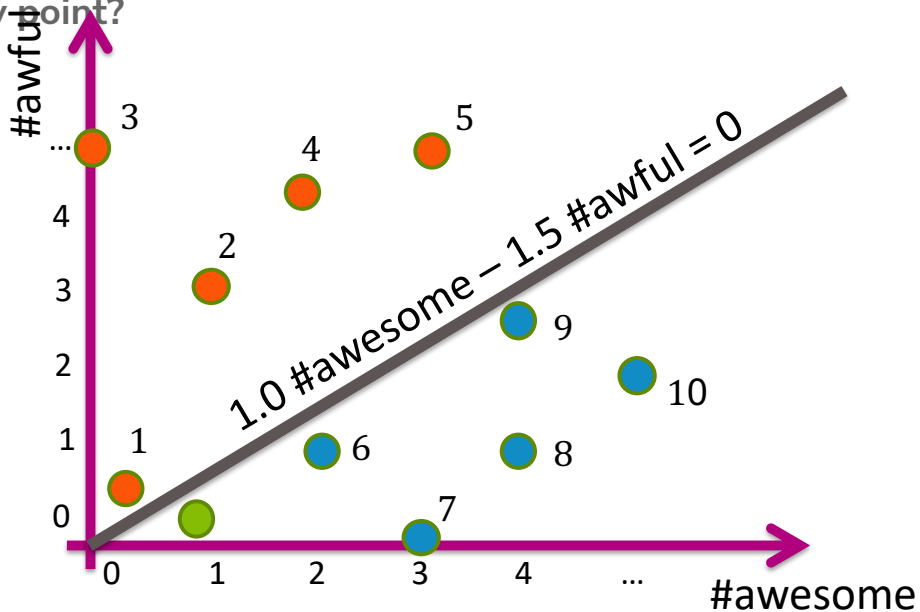
# Poll Everywhere

Think 

1 min

[pollev.com/cs416](http://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...



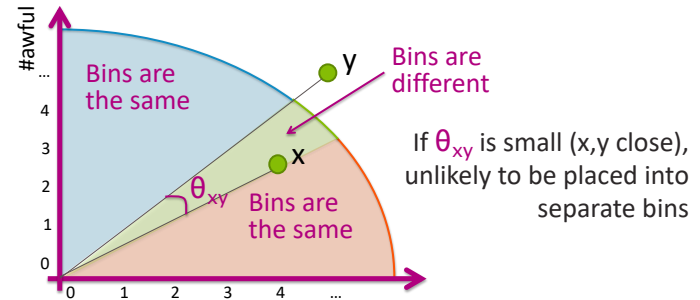
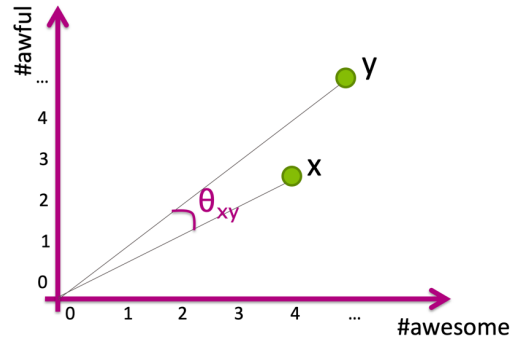
# 1. How to choose line?

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...







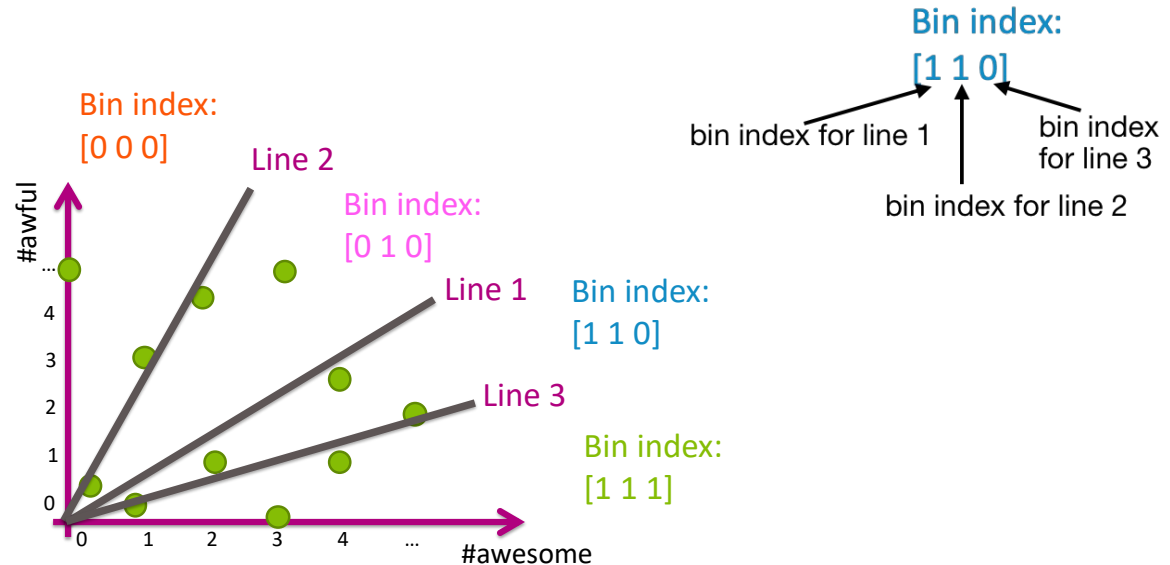
## *Brain Break*



# 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 <sub>1</sub> )	Bin 1 index	Sign (Score <sub>2</sub> )	Bin 2 index	Sign (Score <sub>3</sub> )	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  $x_q$ :

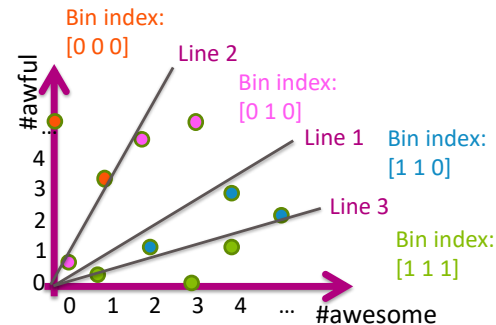
Find its bin index based on the lines

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.

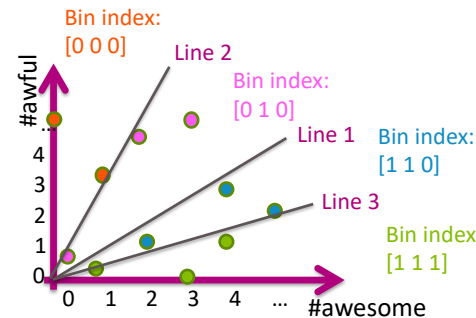
# Improve Quality

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.

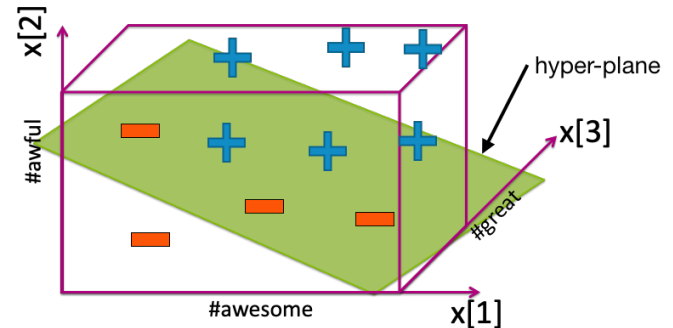
# Higher Dimensions

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  $h \approx \log(d)$

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



# 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



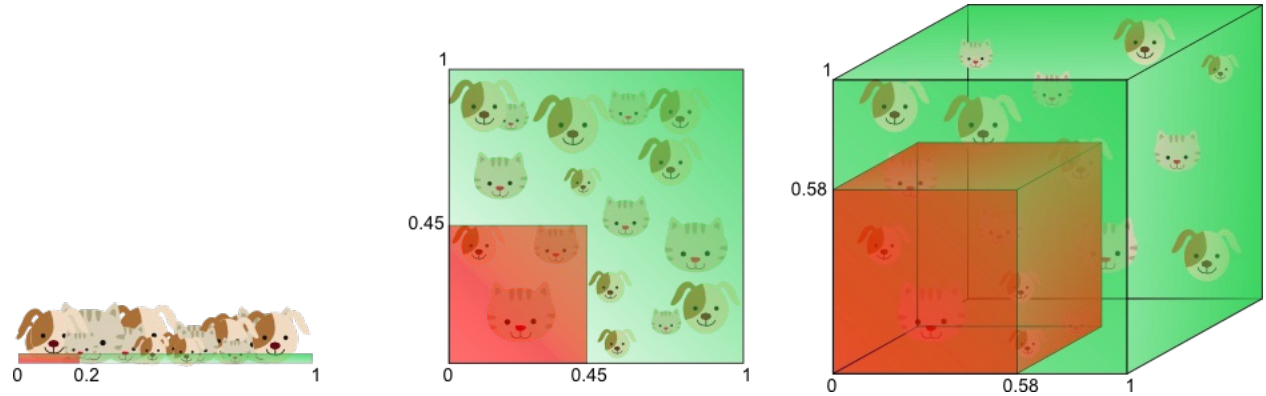


# Curse of Dimensionality

# High Dimensions

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

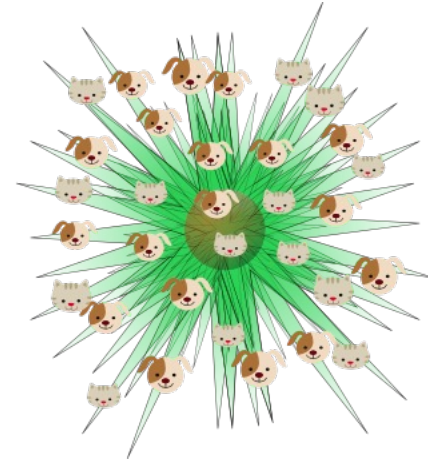
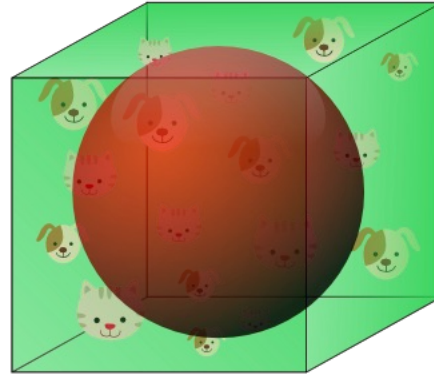
As the number of dimensions grow, the data gets sparser!



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

# Even Weirder

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.



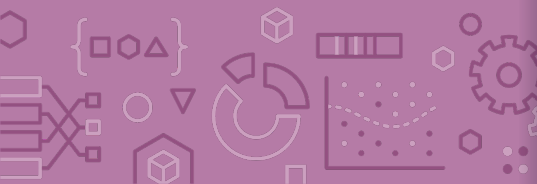
# HW5 – ML Practice with Kaggle

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.

