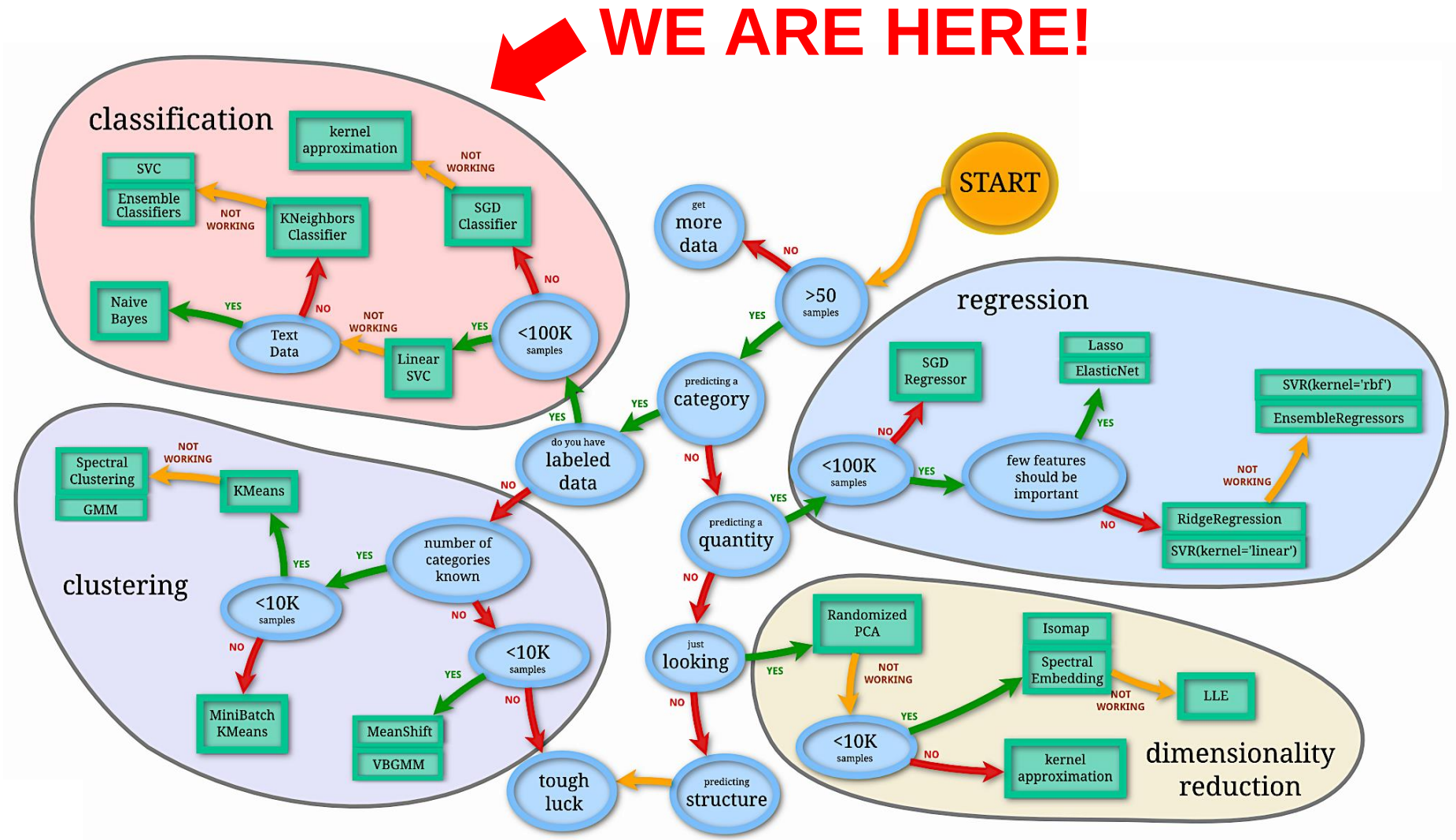
Nearest Neighbor Methods

(Distance-based classification)

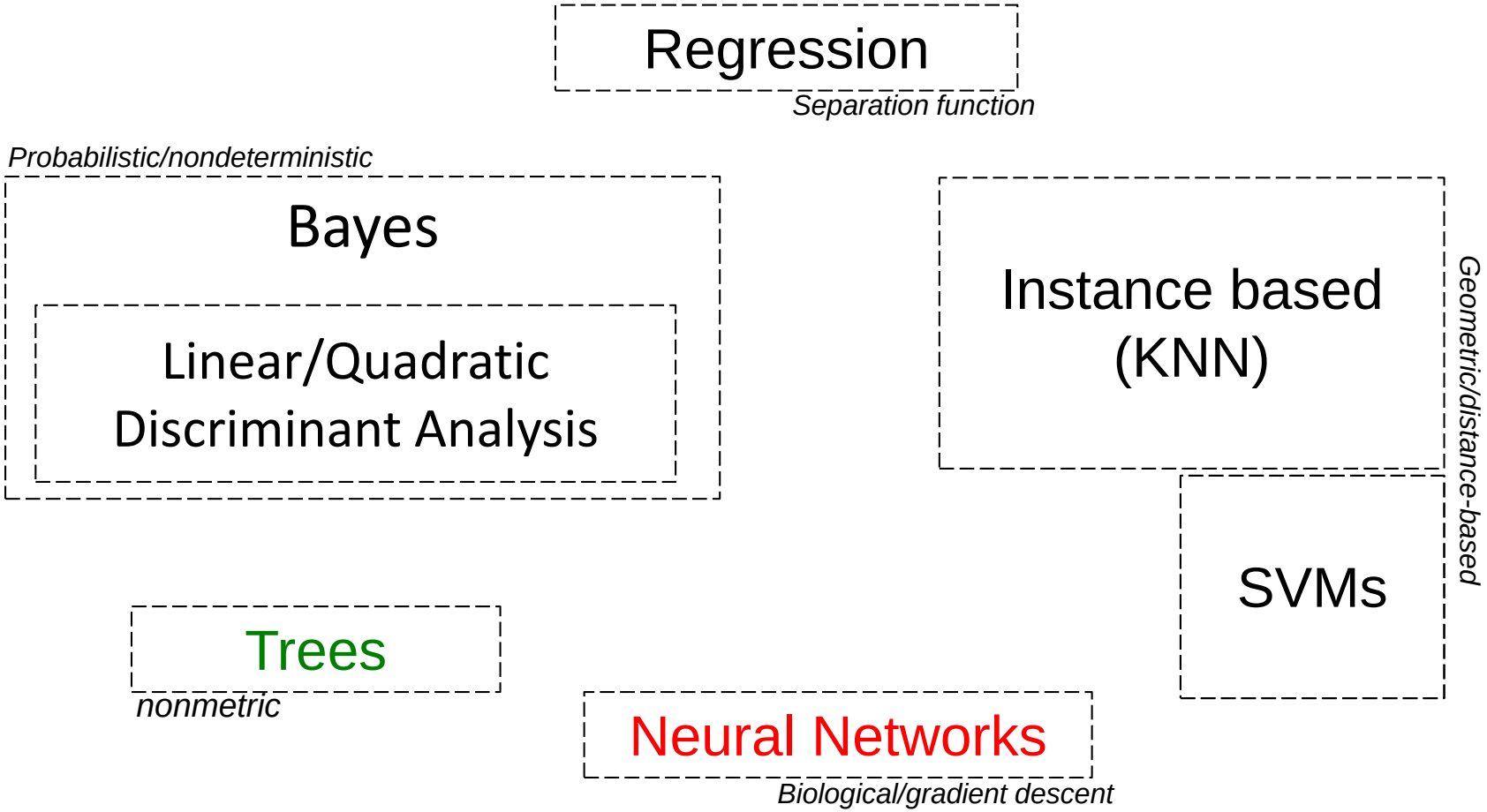
Machine Learning II

Master in Business Analytics and Big Data

acastellanos@faculty.ie.edu


WE ARE HERE!
START
get more data
>50 samples
NO
YES
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
<100K samples
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
<10K samples
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
regression
SGD Regressor
NO
<100K samples
YES
few features should be important
Lasso
ElasticNet
YES
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
dimensionality reduction
Randomized PCA
YES
NOT WORKING
<10K samples
YES
NO
Isomap
Spectral Embedding
NOT WORKING
LLE
kernel approximation

Classifier families (roughly)



"Instance based" classification

- Examples: KNN, RBF, Kernel machines.
- The **idea**: Find an existing instance that is "**most similar**" to the new instance, that we're trying to classify.
  - "Most similar" is a measure of the **distance or similarity** between two instances.
- To classify, select the class/label of (pick one):
  1. The most similar one
  2. The most frequent among the nearest k-neighbors
  3. A weighted majority

Distance/Similarity

The term "***distance***" is really important.

By relying on it, distance-based ML methods find items that are similar based on that "distance" definition. This is, **items that are close to each other**.

Let's see how it works

Distance/Similarity

- A distance is a function that takes two points and returns an scalar value measuring how different those points are

$$d(x,y) \geq 0$$

$$d(x,y) = 0 \quad \text{iif } x=y$$

$$d(x,y) = d(y,x)$$

$$d(x,y) + d(y,z) \geq d(x,z)$$

$$d(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$$

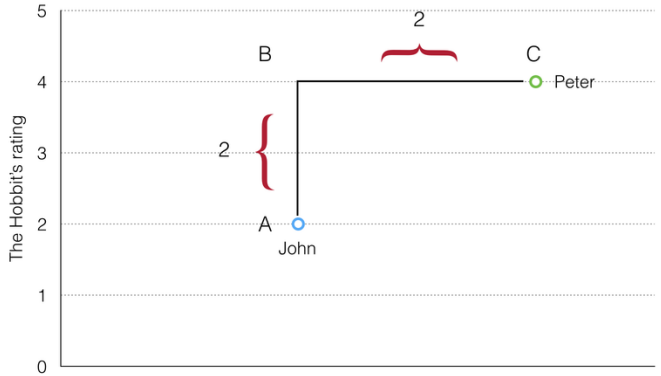
Distance measures

Numerical	Mixable	Manhattan	$d(x,y)=\sum_i^d \lvert x_i - y_i \rvert$
		Euclidean	$d(x,y)=\sqrt{\sum_i^d (x_i - y_i)^2}$
		Maximum	$d(x,y)=\max \lvert x_i - y_i \rvert$
		Minkowski	$d(x,y)=\left(\sum \lvert x_i - y_i \rvert^p\right)^{1/p}$
	Not mixable	Dot product	$x \cdot y$
		Correlation	$\mathrm{corr}(x, y)$
Categorical	Few features		*Hamming distance*
	Binary and Sparse		*Jaccard distance*
Strings	Similar strings		*Hamming distance*
	Very much different		*Levenshtein distance*
			Longest common subsequence
Special purpose	Abstract similarity		*Shared nearest neighbor*

Not mixable: it does not make sense to add a random fraction of one data set to a random fraction of a different data set (e.g.: time series).

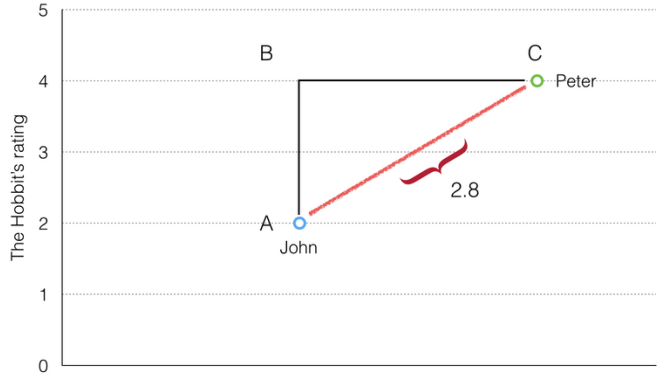
Manhattan Distance

$$d(x,y) = \sum_{k=1}^{n} | x_k - y_k |$$

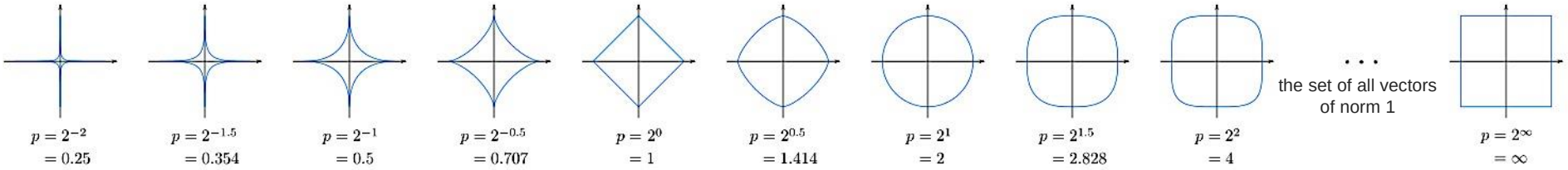


Euclidean Distance

$$d(x,y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

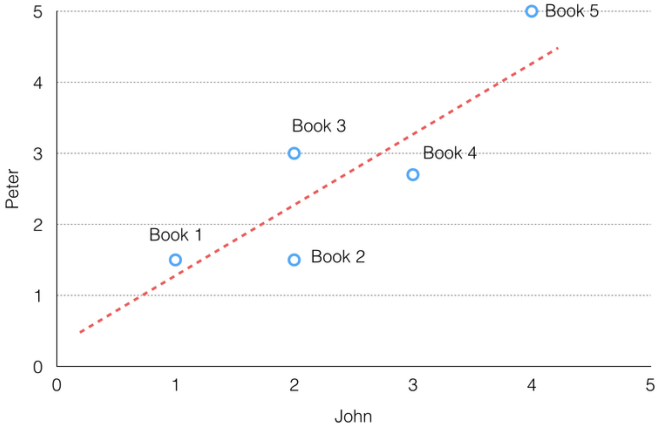


Minkowsky



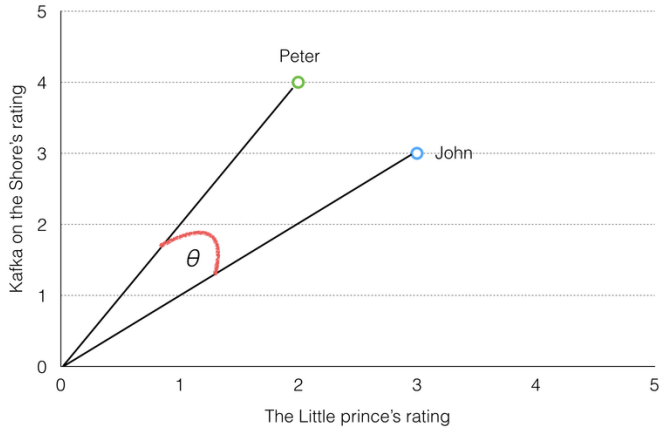
Pearson Correlation Score

$$Pearson(x,y) = \frac{\sum xy - \frac{\sum x \sum y}{N}}{\sqrt{(\sum x^2 - \frac{(\sum x)^2}{N})(\sum y^2 - \frac{(\sum y)^2}{N})}}$$



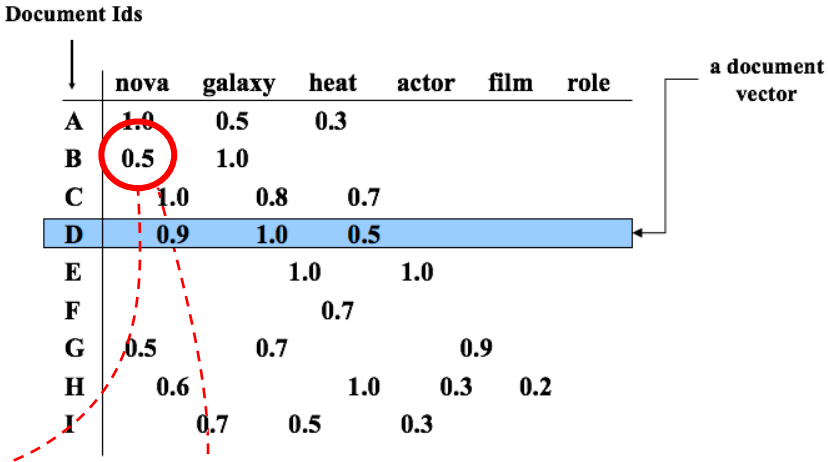
Cosine Distance

$$cos(\theta) = \frac{x \cdot y}{||x|| * ||y||}$$



Example Application #1: Information Retrieval

- **Words are the "axis" of a vector space**
- **Documents are vectors in this space**
  - A vector is an array of floating point (or binary in case of bit maps)
  - Has direction and magnitude
  - Each vector has a place for every term in collection (most are sparse)



options

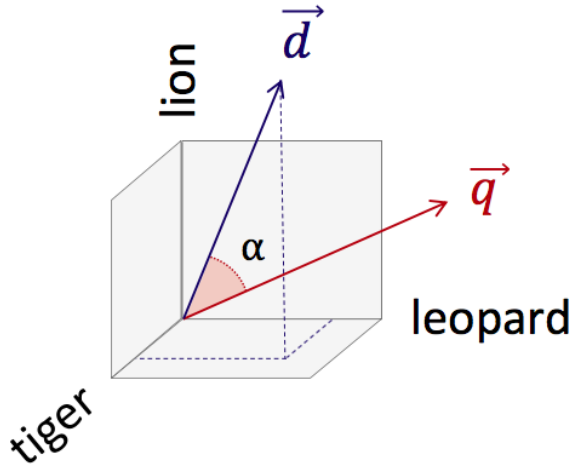
$$\boldsymbol{tf-idf(nova, B)} = 1 + \log_2(\#\ ocurrences\ of\ \text{“}nova\text{”}\ in\ B) \log_2\left(\frac{\#\ docs}{\#docs\ containing\ \text{“}nova\text{”}}\right)$$

Term Frequency / Inverse Document Frequency:
how important a word is to a document in a collection.

$$\boldsymbol{P(nova|B)} \sim \frac{\#\ ocurrences\ of\ \text{“}nova\text{”}\ in\ B}{\#\ words\ in\ B}$$

frequentist

Application #2: Documents & Queries



$$sim(d,q) \equiv \cos(\vec{d},\vec{q})$$

- Documents are represented as vectors in the term space
- Typically values in each dimension correspond to the frequency of the corresponding term in the document
- Queries represented as vectors in the same vector-space
- Cosine similarity between the query and documents is often used to rank retrieved documents

Application #3: Similarities among Documents

- Consider the following document-term matrix

	T1	T2	T3	T4	T5	T6	T7	T8
Doc1	0	4	0	0	0	2	1	3
Doc2	3	1	4	3	1	2	0	1
Doc3	3	0	0	0	3	0	3	0
Doc4	0	1	0	3	0	0	2	0
Doc5	2	2	2	3	1	4	0	2

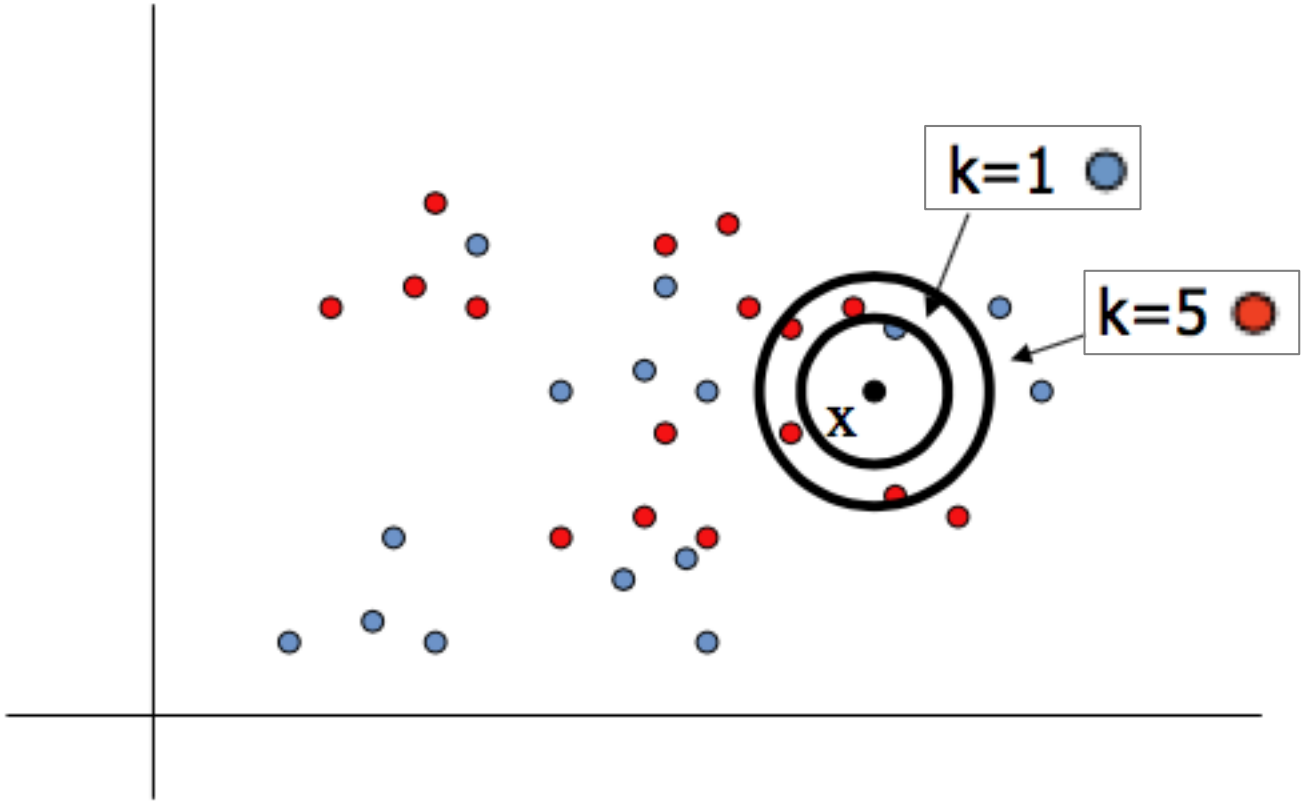
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

```
Dot-Product(Doc2,Doc4) = <3,1,4,3,1,2,0,1> * <0,1,0,3,0,0,2,0>
                                  0 + 1 + 0 + 9 + 0 + 0 + 0 + 0 = 10

Norm (Doc2) = SQRT(9+1+16+9+1+4+0+1) = 6.4
Norm (Doc4) = SQRT(0+1+0+9+0+0+4+0) = 3.74

Cosine(Doc2, Doc4) = 10 / (6.4 * 3.74) = 0.42
```

K-Nearest Neighbor



K-Nearest Neighbor

- We have no knowledge about the distribution of the data (Y), given X.
- So, this method tries to estimate the conditional distribution of Y given X, and then **classify a given observation to the class with highest estimated probability**.

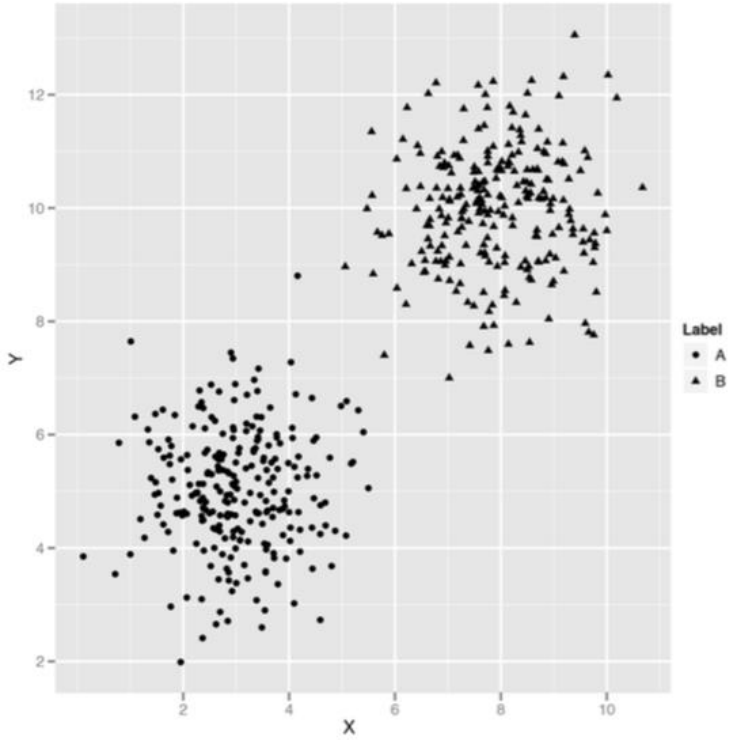
> *The full k-nearest neighbors algorithm works much in the way some of us ask for recommendations from our friends. First, we start with people whose taste we feel we share, and then we ask a bunch of them to recommend something to us. If many of them recommend the same thing, we deduce that we'll like it as well.*
>
> *Machine learning for hackers.*

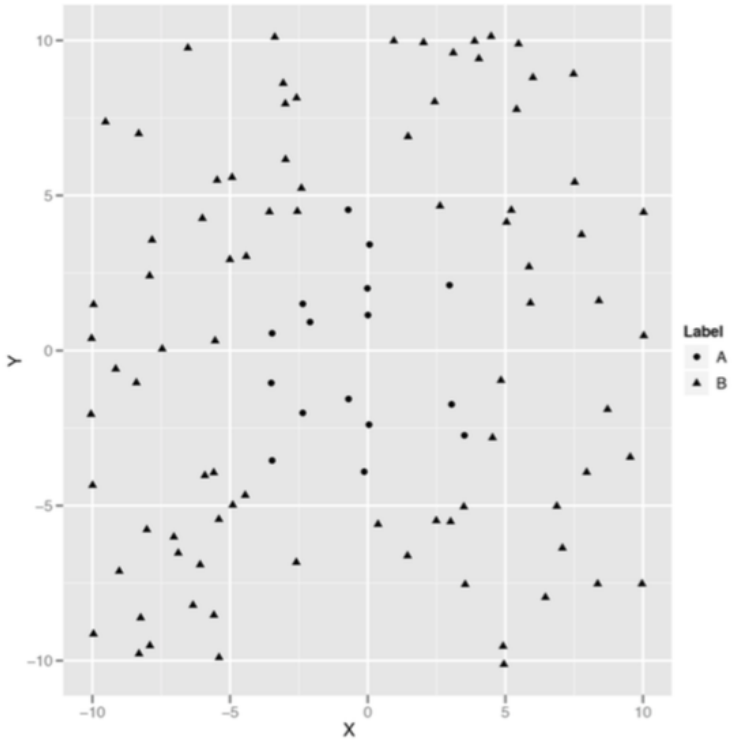
Distance-Based Classification features

- **Memory based**
  - learning by **memorization:** memorize all previously encountered instances.
  - given a new instance, find one **from the memorized set that most closely "resembles"** the new one and **assign new instance to the same class as the "nearest neighbor".**
  - how do we **define "resembles"**? Many options for distance and similarity functions.
- KNN is "**lazy**"
  - defers all of the real work until new instance is obtained; no attempt is made to learn a generalized model from the training set.
  - less data preprocessing and model evaluation, but more work has to be done at classification time.

Decision boundary

How could you try to build a classifier that would match a complicated decision boundary (on the right)?





…use the points nearest the point you're trying to classify to make your guess

Let’s build our own KNN classifier

data

X	Y	Label
2.37354618925767	5.39810588036707	0
3.18364332422208	4.38797360674923	0
2.16437138758995	5.34111969142442	0
4.59528080213779	3.87063690391921	0
3.32950777181536	6.43302370170104	0

...

Algorithm

- Compute the distance between each two samples
- For each ‘item’ in the set of samples:
  1. Retrieve the nearest neighbors (using some distance function) for that ‘item’.
  2. Compute the winning class using majority voting for the ‘k’ closer elements to the ‘item’.
  3. Assign the winning class to the ‘item’

Distance function

- Euclidean distance between each pair of items

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + \cdots + (q_n - p_n)^2}$$

```
distance.matrix <- function(df) {
  distance <- matrix(rep(NA, nrow(df) ^ 2), nrow = nrow(df))
  for (i in 1:nrow(df)) {
    for (j in 1:nrow(df)) {
      distance[i, j] <- sqrt((df[i, 'X'] - df[j, 'X']) ^ 2 + (df[i, 'Y'] - df[j, 'Y']) ^ 2)
    }
  }
  return(distance)
}
```

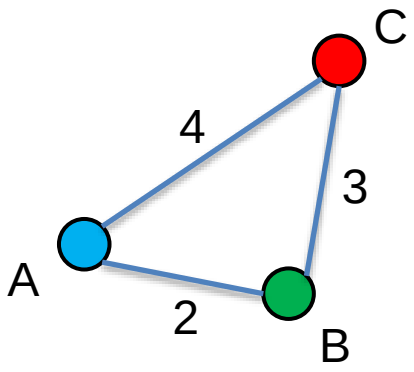
For three points, *a, b, c*, the distance between each pair of points is given by a matrix D:

$$D = \begin{pmatrix} 0 & d(a,b) & d(a,c) \\ d(b,a) & 0 & d(b(c) \\ d(c,a) & d(c,b) & 0 \end{pmatrix}$$

K nearest neighbor

- To access the *k* nearest neighbors of an item (i), we select the $i-th$ row the distance matrix, and reorder in increasing order.

$$D = \begin{pmatrix} 0 & 2 & 4 \\ 2 & 0 & 3 \\ 4 & 3 & 0 \end{pmatrix}$$



The ($k=1$) k nearest neighbors of the 2nd item in our dataset ("B"):

1. We take the 2nd row of the matrix
   $(2 \quad 0 \quad 3)$
2. Reorder it in ascending order
   $(0 \quad 2 \quad 3)$
3. Return only the k first elements
   $(0 \quad \boxed{2} \quad 3)$

We skip the first one, as it is the distance with itself

Nearest neighbor (in R)

```
k.nearest.neighbors <- function(i, distance.matrix, k = 5)
{
  # This just gives us the list of points that are closest to row i
  # in descending order.
  ordered.neighbors <- order(distance.matrix[i, ])

  # The first entry is always 0 (the closest point is the point itself) so
  # let's ignore that entry and return points 2:(k+1) instead of 1:k
  return(ordered.neighbors[2:(k + 1)])
}
```

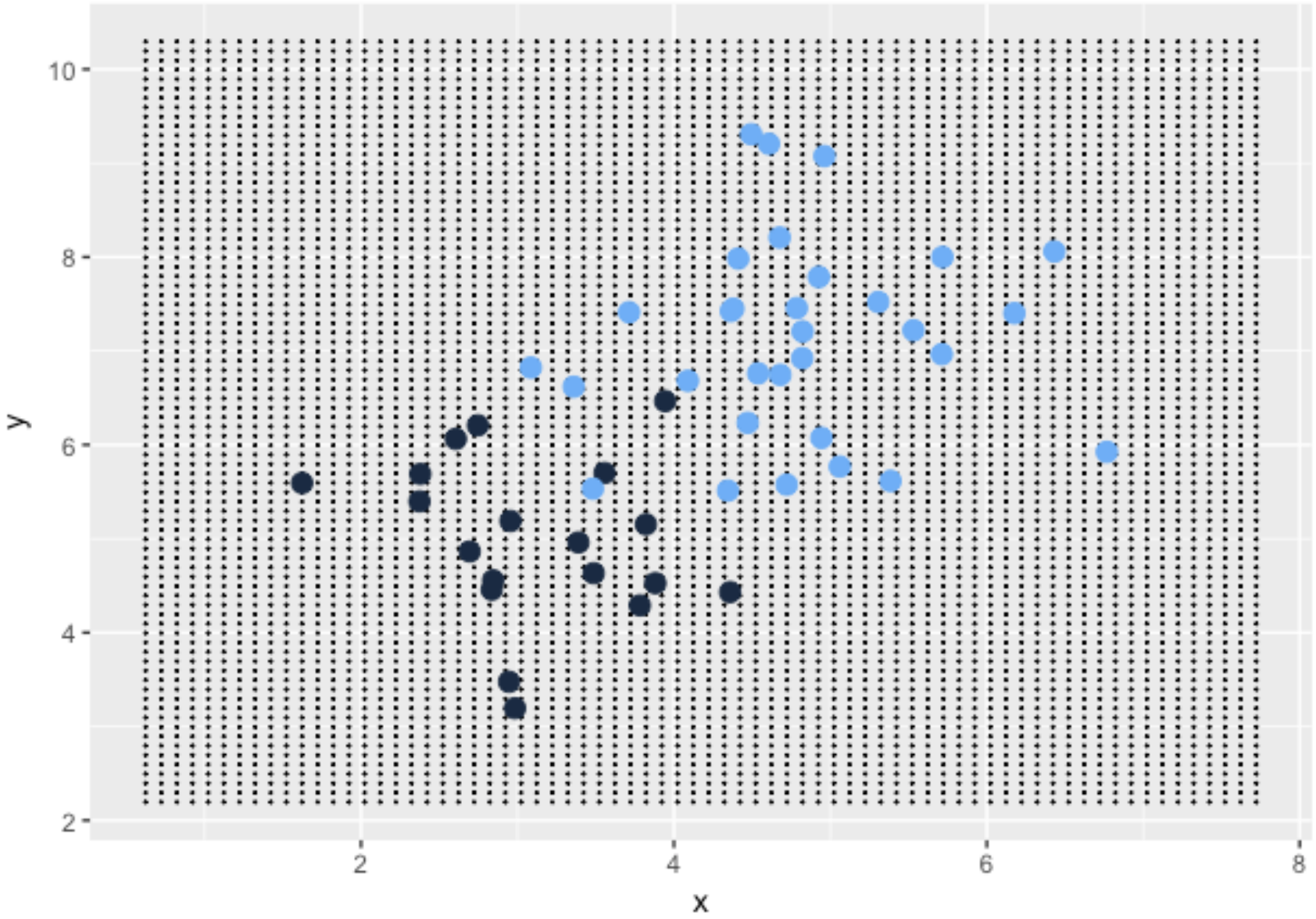
Implementation

- And now, the implementation in R:
  - Majority voting rule to make the predictions, by taking the mean label and then thresholding at 0.5.

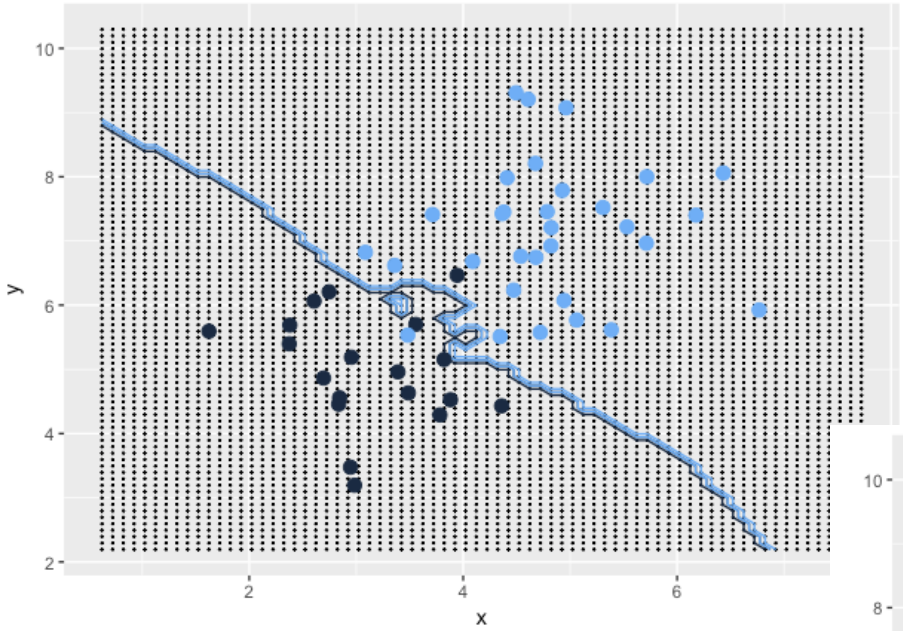
```
knn <- function(mydata, k = 5)
{
  distance <- distance.matrix(mydata)
  predictions <- rep(NA, nrow(mydata))

  for (i in 1:nrow(mydata))
  {
    indices <- k.nearest.neighbors(i, distance, k = k)
    predictions[i] <- ifelse(mean(mydata[indices, 'Label']) > 0.5, 1, 0)
  }

  return(predictions)
}
```

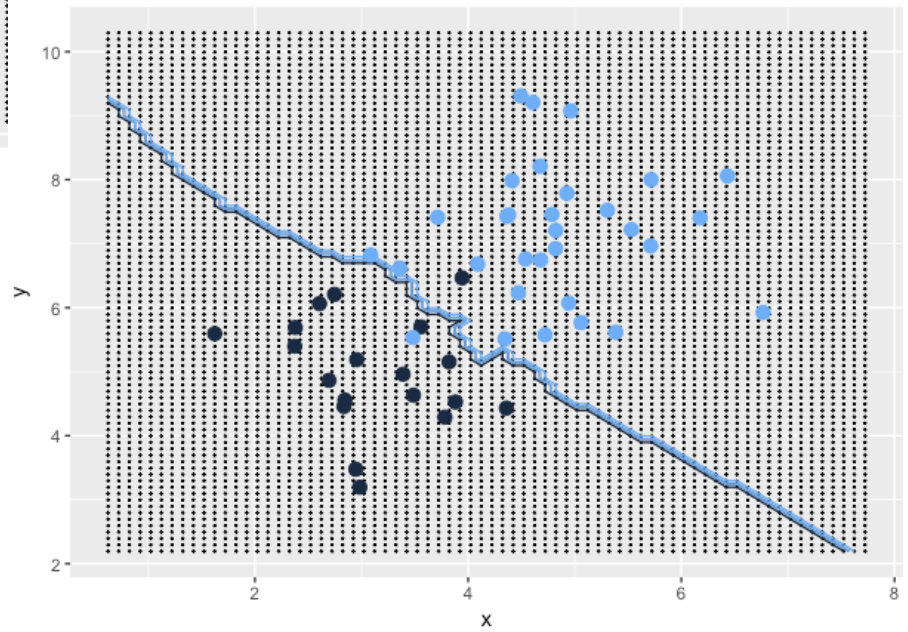

10
8
6
4
2
y
2
4
6
8
x

K = 3



Increasing k reduces variance, increases bias

K = 5



K-Nearest-Neighbor Strategy

- Given object *x*, find the *k* most similar objects to *x*
  - The *k* nearest neighbors
  - **Variety of distance or similarity measures** can be used to identify and rank neighbors
  - Note that this requires comparison between *x* and all objects in the database
- **Classification**:
  - Find the class label for each of the *k* neighbors
  - Use a **voting or weighted** voting approach to determine the majority class among the neighbors (a combination function)
    - Weighted voting means the closest neighbors count more
  - Assign the majority class label to *x*
- **Prediction/Regression**:
  - Identify the value of the target attribute for the *k* neighbors
  - Return the **weighted average as the predicted value** of the target attribute for *x*

Combination Functions

- **Voting**: the "democracy" approach
  - poll the neighbors for the answer and use the majority vote
  - the number of neighbors (*k*) is often taken to be odd in order to avoid ties
    - works when the number of classes is two
    - If there are more than two classes, take k to be the number of classes plus 1

- Impact of *k* on predictions
  - in general different values of *k* affect the outcome of classification
  - we can associate a **confidence level** with predictions (this can be the % of neighbors that are in agreement)
  - problem is that no single category may get a majority vote
  - if there is **strong variations in results for different choices of *k*, this an indication that the training set is not large enough**

Voting Approach - Example

Will a new customer respond to solicitation?

ID	Gender	Age	Salary	Respond?
1	F	27	19,000	no
2	M	51	64,000	yes
3	M	52	105,000	yes
4	F	33	55,000	yes
5	M	45	45,000	no
new	F	45	100,000	?

Using the voting method without confidence

	Neighbors	Answers	k =1	k = 2	k = 3	k = 4	k = 5
D_man	4,3,5,2,1	Y,Y,N,Y,N	yes	yes	yes	yes	yes
D_euclid	4,1,5,2,3	Y,N,N,Y,Y	yes	?	no	?	yes

Using the voting method with a confidence

	k =1	k = 2	k = 3	k = 4	k = 5
D_man	yes, 100%	yes, 100%	yes, 67%	yes, 75%	yes, 60%
D_euclid	yes, 100%	yes, 50%	no, 67%	yes, 50%	yes, 60%

Combination Functions

- **Weighted Voting**: not so "democratic"
  - similar to voting, but the vote of some neighbors counts more
  - question is: which neighbor's vote counts more?
- **How can weights be obtained?**
  - **Distance-based**
    - closer neighbors get higher weights
    - "value" of the vote is the inverse of the distance (may need to add a small constant)
    - the weighted sum for each class gives the combined score for that class
    - to compute confidence, need to take weighted average
  - **Heuristic**
    - weight for each neighbor is based on domain-specific characteristics of that neighbor
- **Advantage of weighted voting**
  - introduces enough variation to prevent ties in most cases
  - helps distinguish between competing neighbors

*K*NN and Collaborative Filtering

- Collaborative Filtering Example
  - A movie rating system
  - Ratings scale: 1 = “hate it”; 7 = “love it”
  - Historical DB of users includes ratings of movies by Sally, Bob, Chris, and Lynn
  - Karen is a new user who has rated 3 movies, but has not yet seen “Independence Day”; should we recommend it to her?

Will Karen like “Independence Day?”

	Sally	Bob	Chris	Lynn	Karen
Star Wars	7	7	3	4	7
Jurassic Park	6	4	7	4	4
Terminator II	3	4	7	6	3
Independence Day	7	6	2	2	?

Collaborative Filtering (kNN Example)

	Star Wars	Jurassic Park	Terminator 2	Indep. Day	*Average*	Cosine	Distance	Euclid	Pearson
Sally	7	6	3	7	5.33	0.983	2	2.00	0.85
Bob	7	4	4	6	5.00	0.995	1	1.00	0.97
Chris	3	7	7	2	5.67	0.787	11	6.40	-0.97
Lynn	4	4	6	2	4.67	0.874	6	4.24	-0.69
Karen	7	4	3	?	4.67	1.000	0	0.00	1.00

K	Prediction
1	6
2	6.5
3	5

K is the number of nearest neighbors used in to find the average predicted ratings of Karen on Indep. Day.

Correlation:

$$Pearson(Sally, Karen) = 0.85$$

Summary of KNN

- To make a prediction, **KNN identifies those K training observations that are closest to X.**
- And then, **X is assigned to the class to which the plurality of these observations belong**.
- Therefore
  - KNN is superior to LDA or Logistic Regression when the decision boundary is highly non-linear
  - However, KNN does not tell anything about what predictors are important.

Summary of KNN (cont.)

- **Interpretation:**
  - The whole model is difficult to interpret beyond the mere similarity between neighbors.
  - This makes KNN appropriate for recommendation systems
- **Dimensionality/efficiency**
  - KNN considers all features (predictors) in the problem space.
  - **Solution:** perform **feature selection** or inject domain knowledge into similarity calculation by setting your own **distance function**.

Summary of KNN (cont.)

- **When to use it?**
  - Less than 20 features
  - Lots of training data
  - Always scale/standardize feature values.
  - Training is very fast
  - Learn complex target functions
- **Advantages**
  - Easy to program
  - No optimization or training required
  - Classification accuracy can be very good; can outperform more complex models
- **Disadvantages**
  - Slow at query time
  - Easily fooled by irrelevant attributes