

INTRODUCTION TO DATA SCIENCE

Lecture #22

CMSC320



COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

k-NN

First let's define a *neighbor*

k -NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*

k -NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:

k -NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
 - 2.1 Is B or C closer to point A ?

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
 - 2.1 Is *B* or *C* closer to point *A*?
3. We can ask this for every point in the dataset, for *every other* point in the dataset

k-NN

First let's define a *neighbor*

1. For many types of features, we can have a notion of *distance*
2. Using that notion of distance we can ask questions like:
 - 2.1 Is *B* or *C* closer to point *A*?
3. We can ask this for every point in the dataset, for *every other* point in the dataset
4. This can give us an *ordering* of the nearest neighbors.

k...

$k\dots$

1. When $k = 1$, we only care about the closet point.

$k\dots$

1. When $k = 1$, we only care about the closet point.
2. When $k = 2$, we only care about the two closet points, and they ‘vote’.

$k\dots$

1. When $k = 1$, we only care about the closet point.
2. When $k = 2$, we only care about the two closet points, and they ‘vote’.
3. When $k = 3$, we only care about the... you get it.

Discussion time.

What do we have to do in order to train this model?

Discussion time.

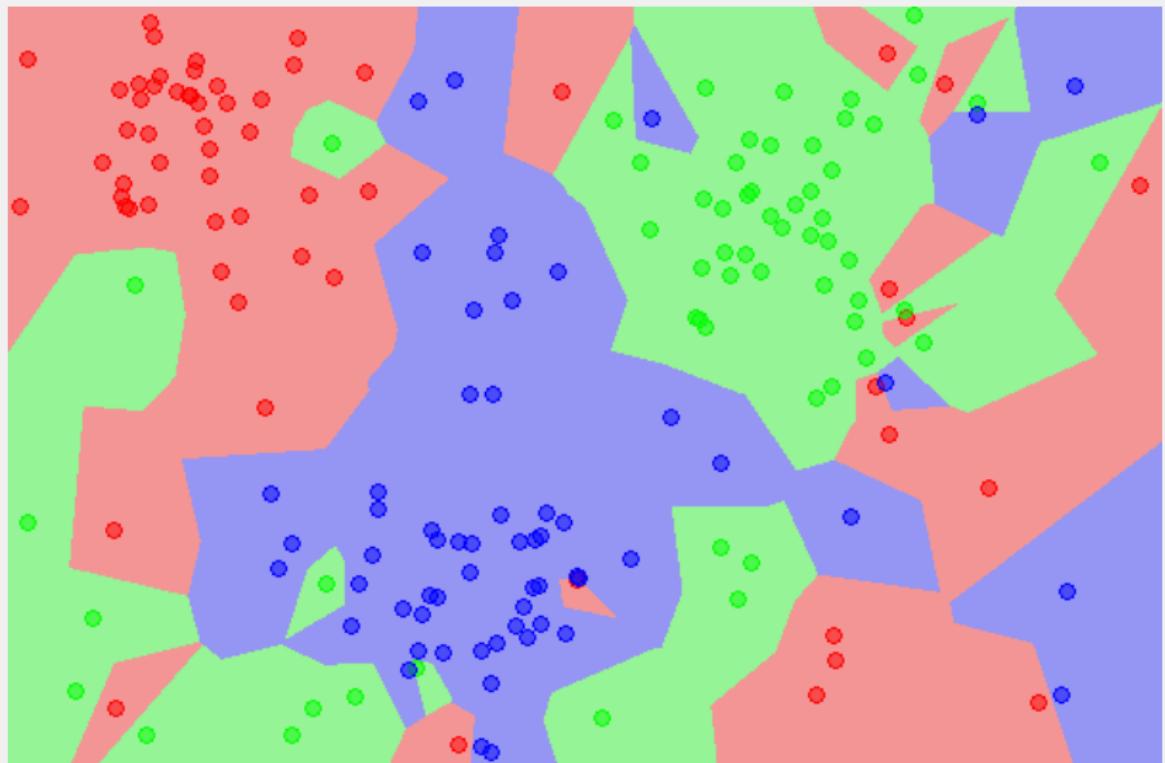
What do we have to do in order to train this model?

1. We only have to store the dataset in a way that let's us calculate the nearest neighbor(s)

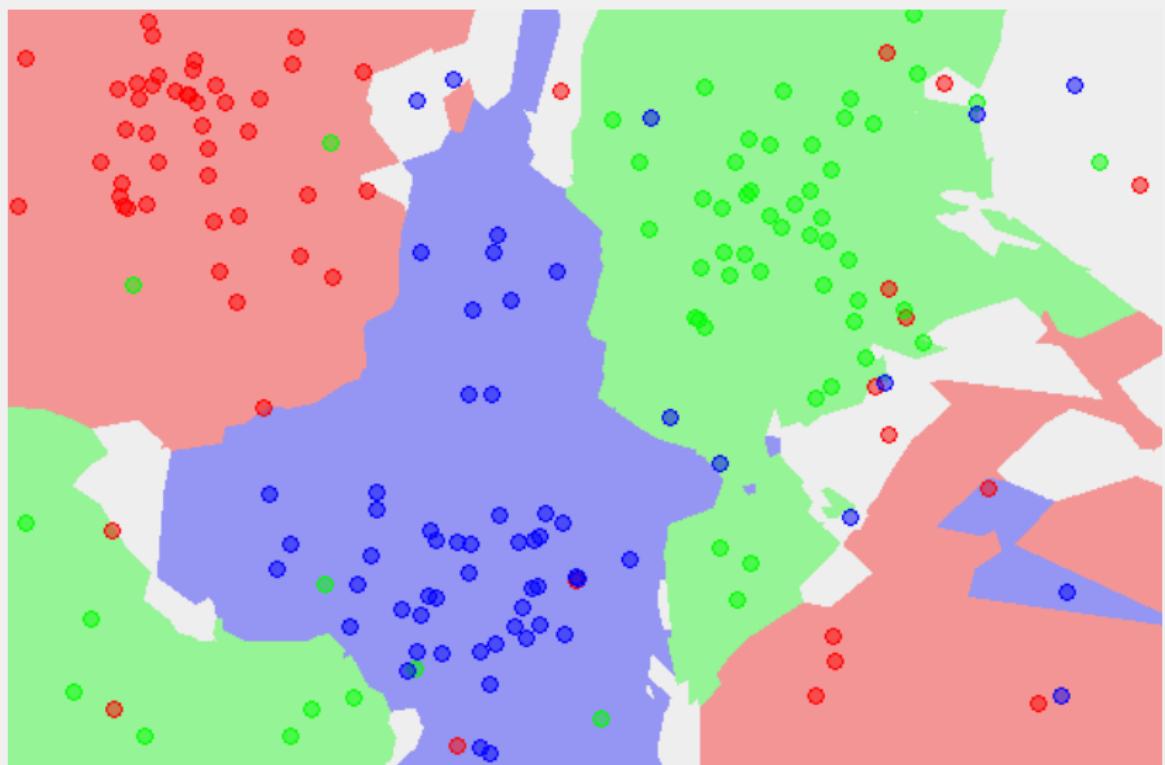
Discussion time 2.

What do we think k -NN might be good for?

Let's take a look (1NN):



Let's take a look (5NN):



k-NN and SKLearn

You might see a pattern developing:

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`
2. Defaults to $k = 5$

k-NN and SKLearn

You might see a pattern developing:

1. `sklearn.neighbors.KNeighborsClassifier`
2. Defaults to $k = 5$
3. You can also play with the `metric` and `p` parameters to change how distance is calculated (default is Euclidean distance).

Radial basis functions (RBFs)

For $x \in \mathbb{R}^k$, select some set of p centers, $\mu^{(1)}, \dots, \mu^{(p)}$ (we'll discuss shortly how to select these), and create features

$$\phi = \left\{ \exp\left(-\frac{\|x - \mu^{(i)}\|_2^2}{2\sigma^2}\right) : i = 1, \dots, p \right\} \cup \{1\} \in \mathbb{R}^{p+1}$$

Very important: need to normalize columns of X (i.e., different features), to all be the same range, or distances wont be meaningful

(Hyper)parameters of the features include the choice of the p centers, and the choice of the *bandwidth* σ

Choose centers, i.e., to be a uniform grid over input space, can choose σ e.g. using cross validation (don't do this, though, more on this shortly)

Example radial basis function

Example:

$$x = [\text{High} - \text{Temperature}],$$

$$\mu^{(1)} = [20], \mu^{(2)} = [25], \dots, \mu^{(16)} = [95], \sigma = 10$$

Leads to features:

$$\phi = \begin{bmatrix} \exp(-(x - 20)^2 / 200) \\ \vdots \\ \exp(-(x - 95)^2 / 200) \\ 1 \end{bmatrix}$$

Code for generating RBFs

The following code generates a complete set of RBF features for an entire data matrix $X \in \mathbb{R}^{m \times k}$ and matrix of centers $\mu \in \mathbb{R}^{p \times k}$

```
def rbf(X, mu, sig):
    sqdist = (-2*X.dot(mu.T) +
              np.sum(X**2, axis=1)[ :, None ] +
              np.sum(mu**2, axis=1))
    return np.exp(-sqdist/(2*sig**2))
```

Important “trick” is to efficiently compute distances between *all* data points and all centers

Difficulties with general features

The challenge with these general non-linear features is that the number of potential features grows very quickly in the dimensionality of the raw input

Polynomials: k -dimensional raw input $\Rightarrow \binom{k+d}{k} = O(d^k)$ total features (for fixed d)

RBFs: k -dimensional raw input, uniform grid with d centers over each dimension $\Rightarrow d^k$ total features

These quickly become impractical for large feature raw input spaces

Practical RBFs

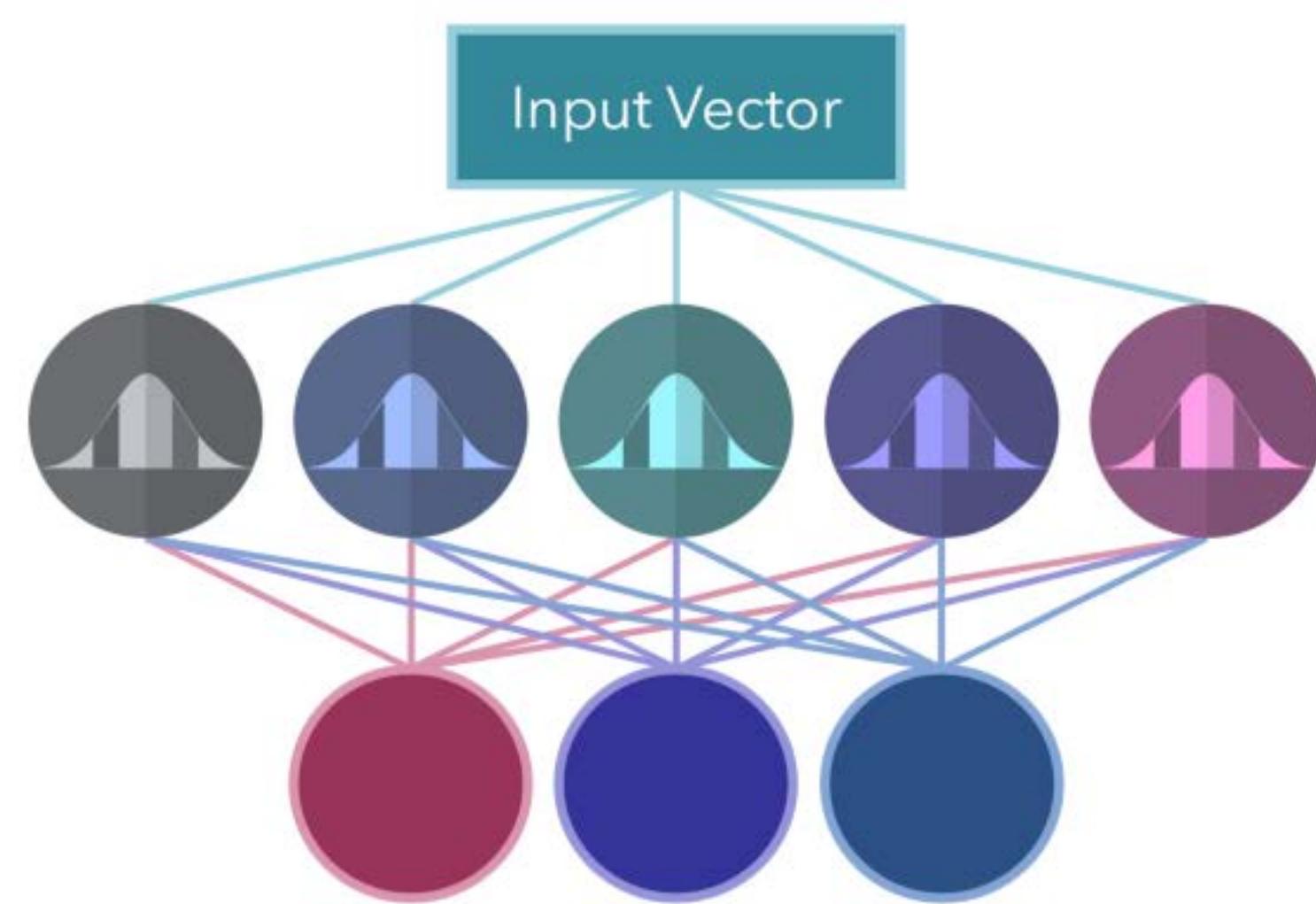
Don't create RBF centers in a grid over your raw input space (your data will never cover an entire high-dimensional space, but will lie on a subset)

Instead, pick centers by randomly choosing p data points in the training set (a bit fancier, run k-means to find centers, which we'll describe later)

Don't pick σ using cross validation

Instead, choose the following (called the *median trick*)

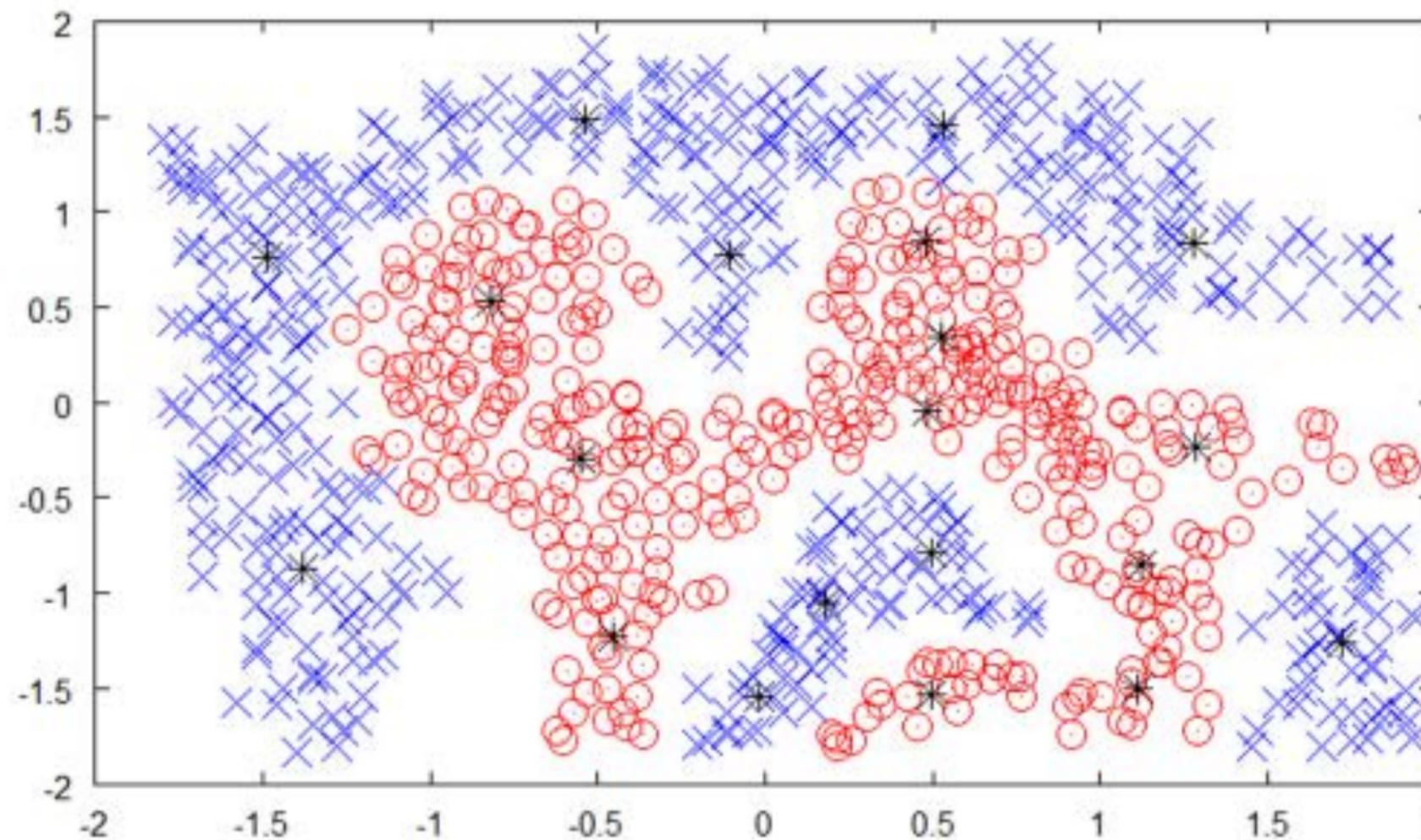
$$\sigma = \text{median}(\{\|\mu^{(i)} - \mu^{(j)}\|_2, i, j = 1, \dots, p\})$$

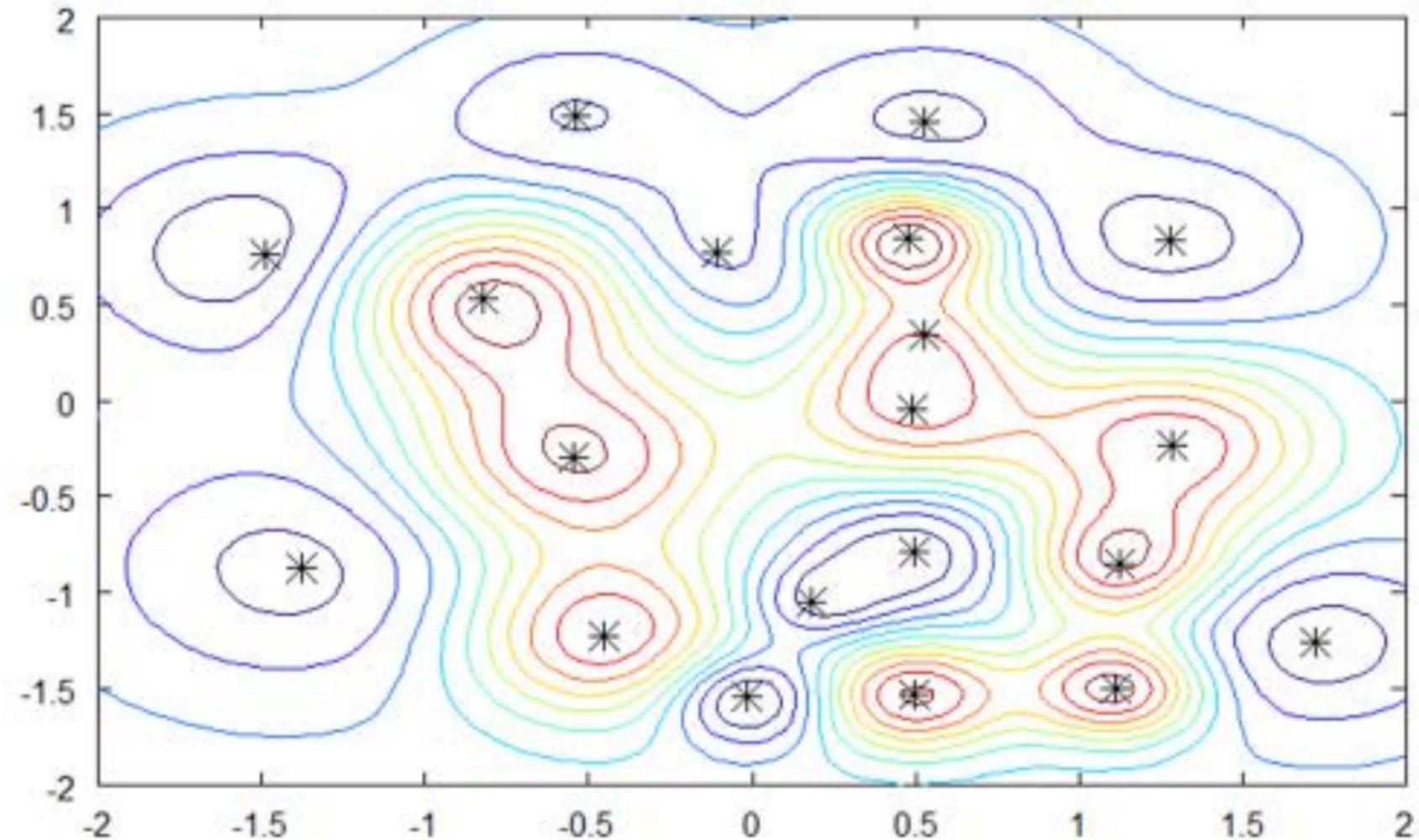


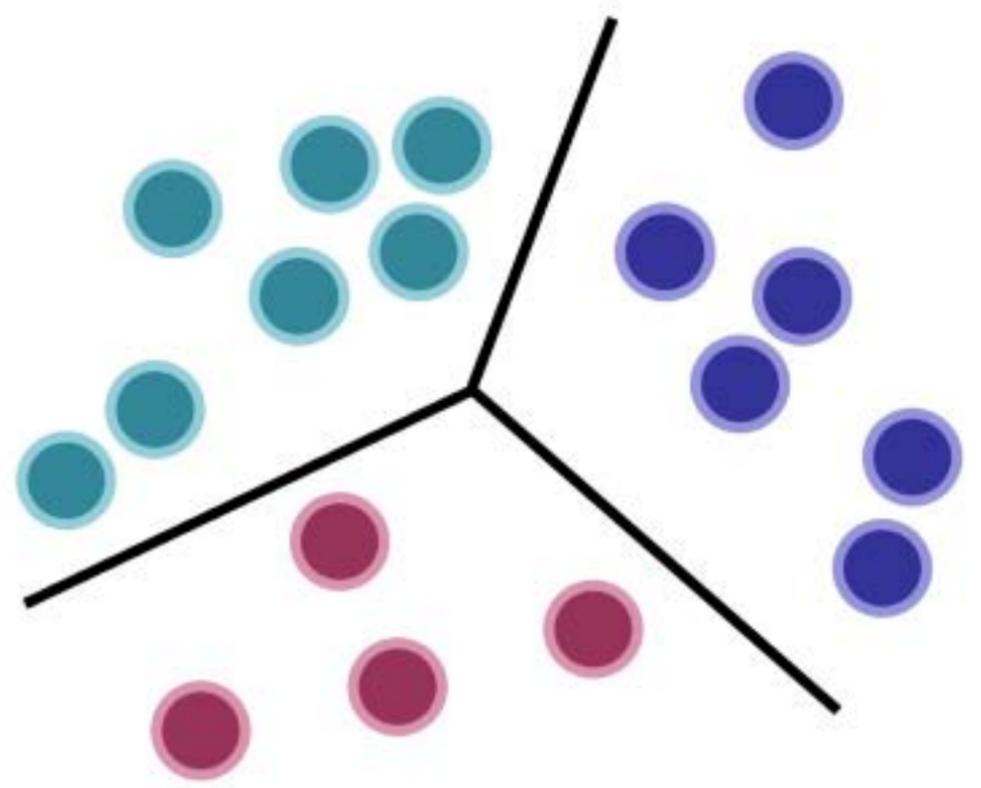
The input vector

Radial basis neurons

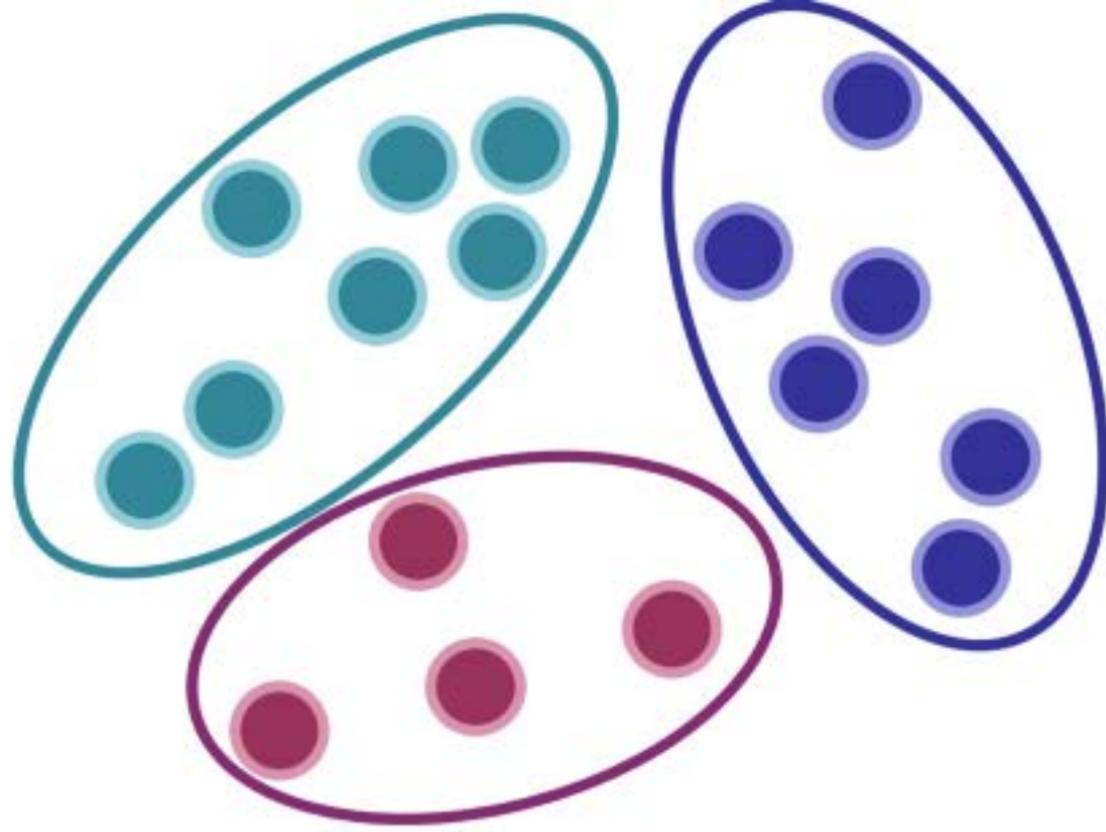
Output neurons
(weighted sums)







Standard Neural Network

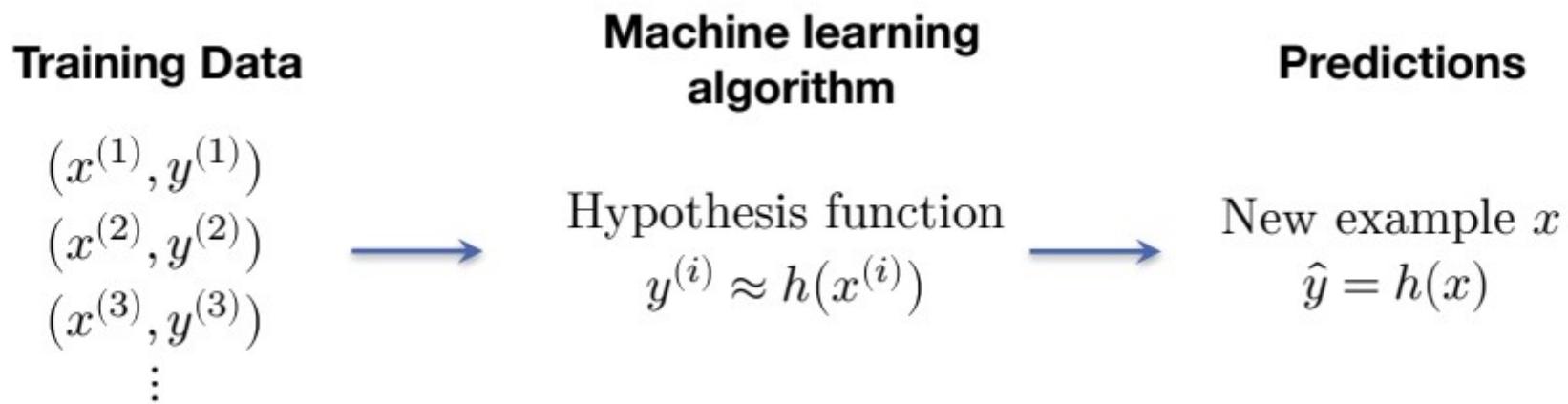


Radial Basis Network

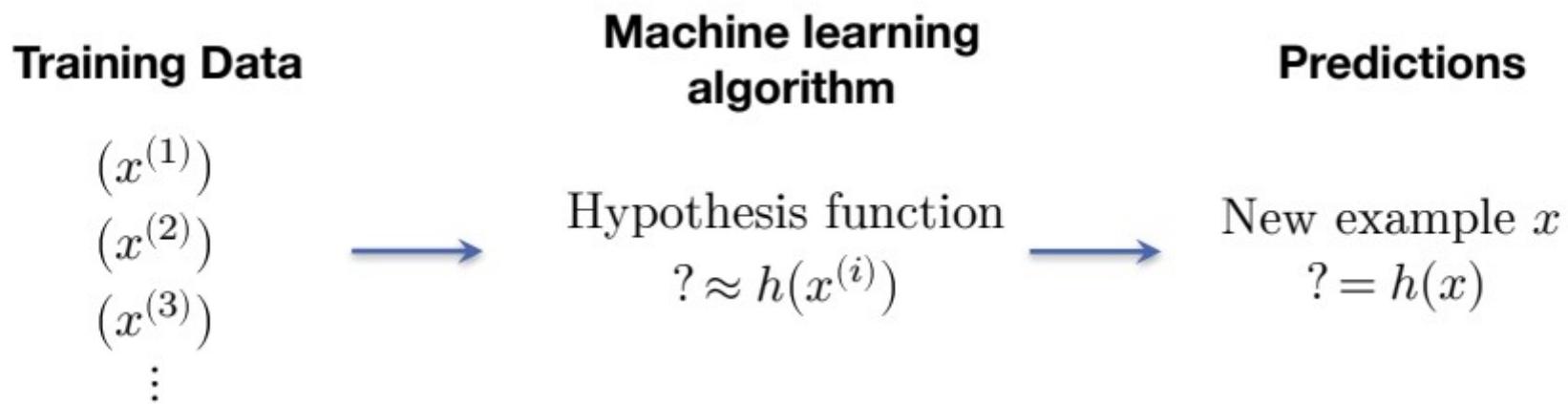
A group of diverse cartoon characters from the TV show "Bob's Burgers" are standing together outdoors at night. The characters include Bob Belcher, Linda Belcher, Tina Belcher, Louise Belcher, Mr. Belcher, Gene Belcher, and other townspeople like Mr. Hanksy and Mrs. Hanksy. They are all looking towards the right side of the frame.

Unsupervised LEARNING

Supervised learning paradigm



Unsupervised learning paradigm



Three elements of unsupervised learning

It turns out the virtually all unsupervised learning algorithms can be considered in the same manner as supervised learning:

1. Define hypothesis function
2. Define loss function
3. Define how to optimize the loss function

But, what do a hypothesis function and loss function signify in the unsupervised setting?

UNSUPERVISED LEARNING

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \dots, m$

Model parameters: $\theta \in \mathbb{R}^k$

Hypothesis function: $h_\theta: \mathbb{R}^n \rightarrow \mathbb{R}^n$??????????????

Want: approximate input given input, or $x^{(i)} \approx h_\theta(x^{(i)})$

Loss function: $\ell: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}_+$

$$h_\theta(x) = \operatorname{argmin}_{\mu \in \{\mu^{(1)}, \dots, \mu^{(k)}\}} \|\mu - x\|_2^2 \quad \ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

$$\operatorname{minimize}_\theta \sum_{i=1}^m \ell(h_\theta(x^{(i)}), x^{(i)})$$

Hypothesis and loss functions

The framework seems odd, what does it mean to have a hypothesis function approximate the input?

Can't we just pick $h_\theta(x) = x$?

The goal of unsupervised learning is to pick some *restricted* class of hypothesis functions that extract some kind of structure from the data (i.e., one that does not include the identity mapping above)

In this lecture, we'll consider two different algorithms that both fit the framework: k-means and principal component analysis

K-means graphically

The k-means algorithm is easy to visualize: given some collection of data points we want to find k centers such that all points are close to at least one center



K-means in unsupervised framework

Parameters of k-means are the choice of centers $\theta = \{\mu^{(1)}, \dots, \mu^{(k)}\}$, with $\mu^{(i)} \in \mathbb{R}^n$

Hypothesis function outputs the center closest to a point x

$$h_\theta(x) = \operatorname{argmin}_{\mu \in \{\mu^{(1)}, \dots, \mu^{(k)}\}} \|\mu - x\|_2^2$$

Loss function is squared error between input and hypothesis

$$\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

Optimization problem is thus

$$\underset{\mu^{(1)}, \dots, \mu^{(k)}}{\text{minimize}} \sum_{i=1}^m \|h_\theta(x^{(i)}) - x^{(i)}\|_2^2$$

K-MEANS

Non-convex optimization problem ↗ locally good solutions

Given: dataset $x^{(i)}$, number of clusters k

Initialize k cluster centers:

$$\mu^{(j)} \leftarrow \text{Random}(x^{(i)}), \quad j = 1, \dots, k$$

Repeat until convergence or bored:

1. Compute cluster assignments:

$$y^{(i)} = \underset{j}{\operatorname{argmin}} \|\mu^{(j)} - x^{(i)}\|_2^2, \quad i = 1, \dots, m$$

2. Re-compute the new cluster means:

$$\mu^{(j)} \leftarrow \text{Mean}(\{x^{(i)} | y^{(i)} = j\}), \quad j = 1, \dots, k$$

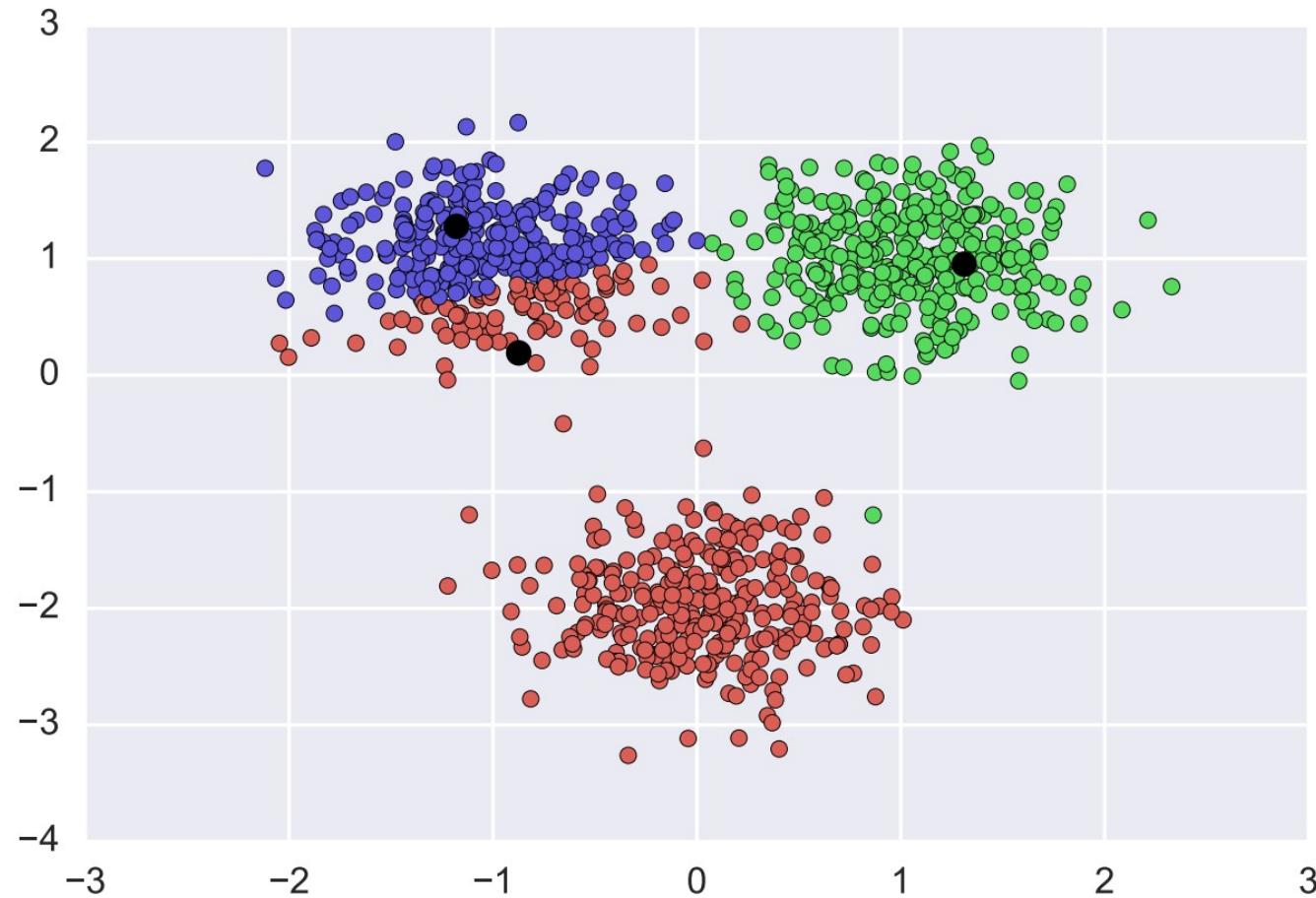
K-means in a few lines of code

Scikit-learn, etc, contains k-means implementations, but again these are pretty easy to write

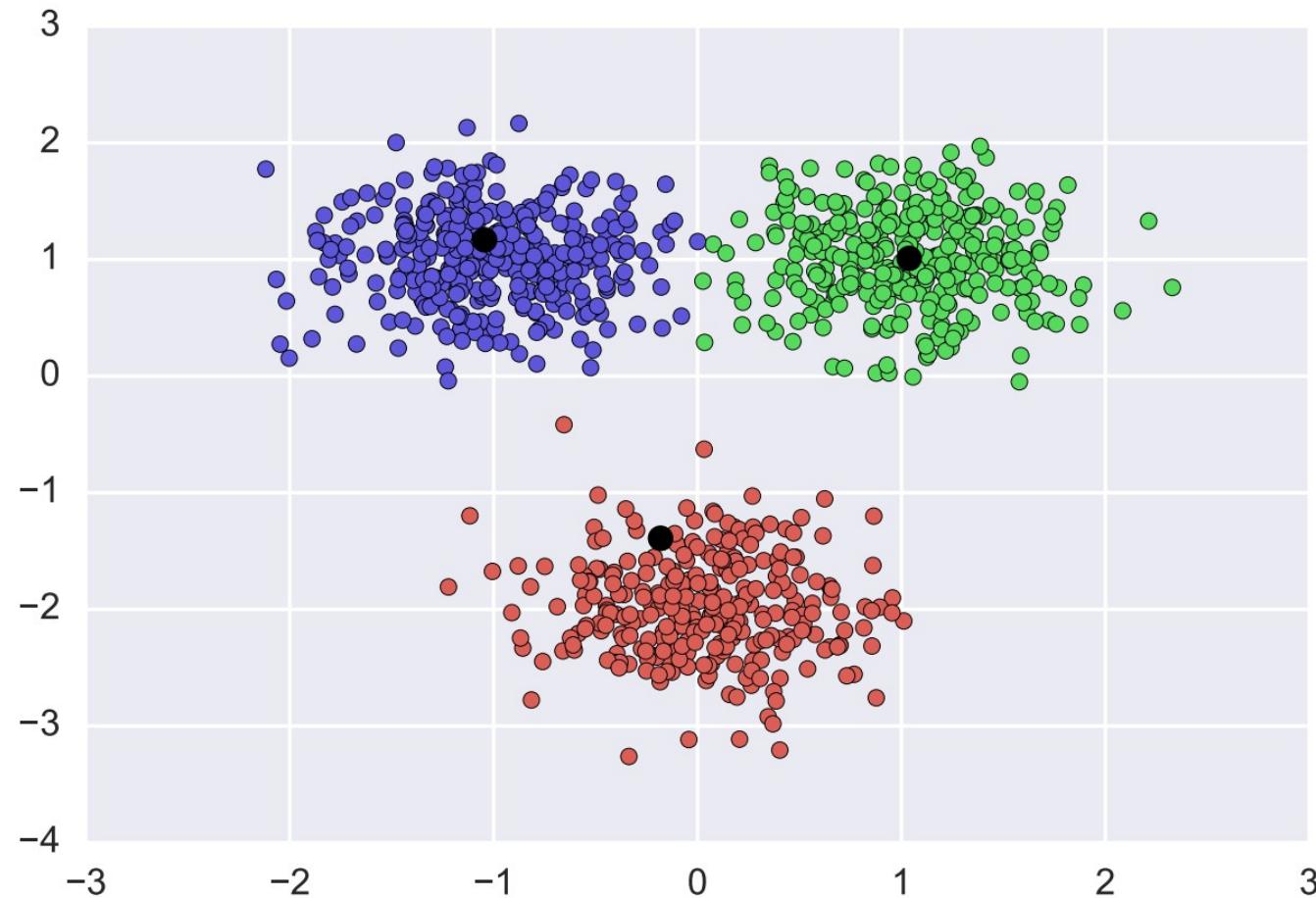
For better implementation, want to check for convergence as well as max number of iterations

```
def kmeans(X, k, max_iter=10):
    Mu = X[np.random.choice(X.shape[0],k),:]
    for i in range(max_iter):
        D = (-2*X.dot(Mu.T) + np.sum(X**2, axis=1)[:,None] +
              np.sum(Mu**2, axis=1))
        C = np.eye(k)[np.argmin(D, axis=1),:]
        Mu = C.T.dot(X)/np.sum(C, axis=0)[:,None]
    loss = np.linalg.norm(X - Mu[np.argmin(D, axis=1),:]**)2
    return Mu, C, loss
```

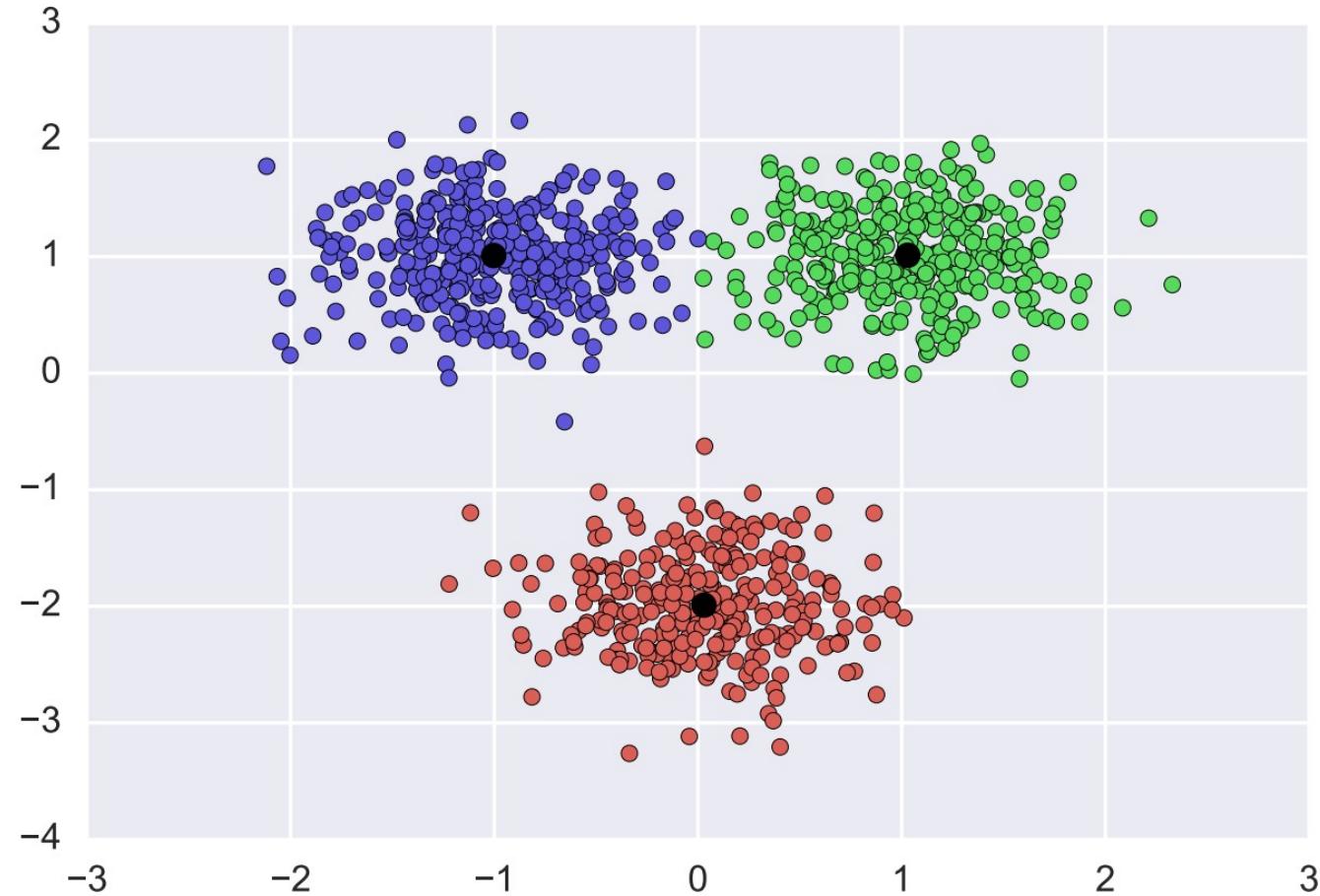
A (LUCKY) EXAMPLE



A (LUCKY) EXAMPLE



A (LUCKY) EXAMPLE

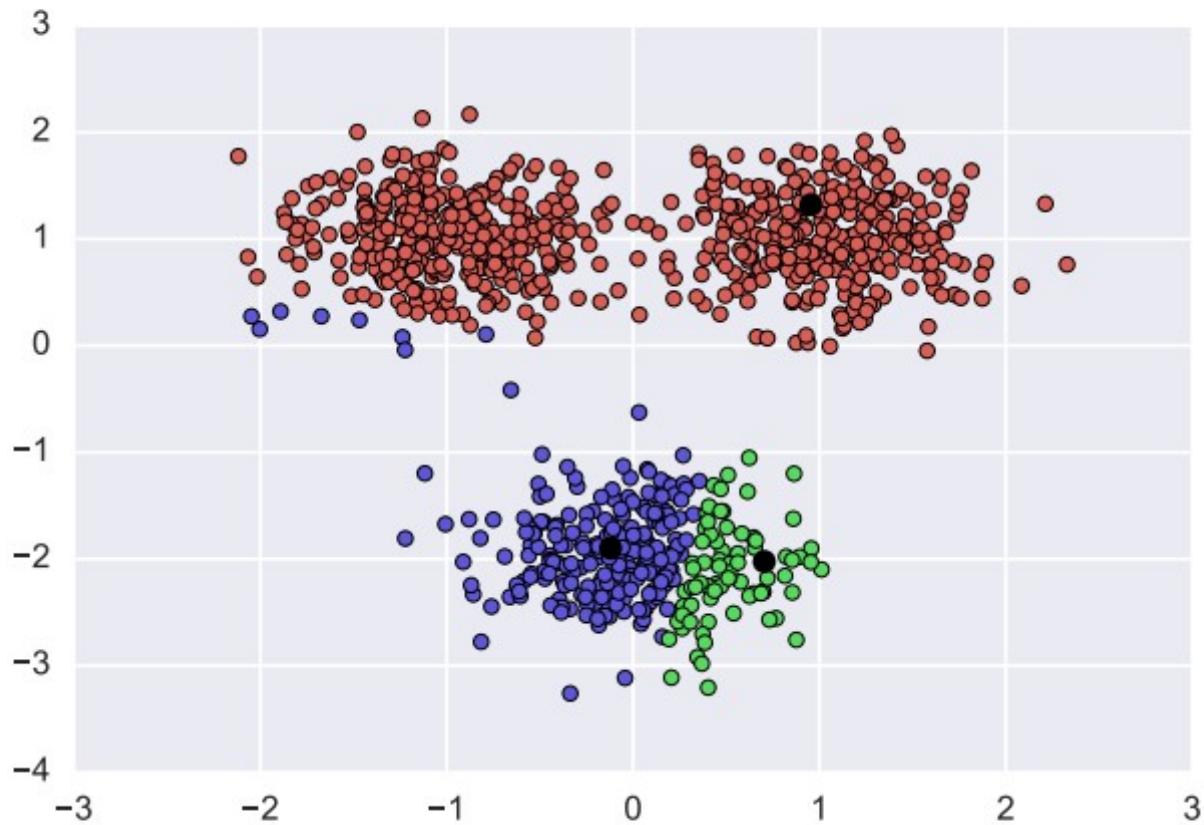


Possibility of local optima

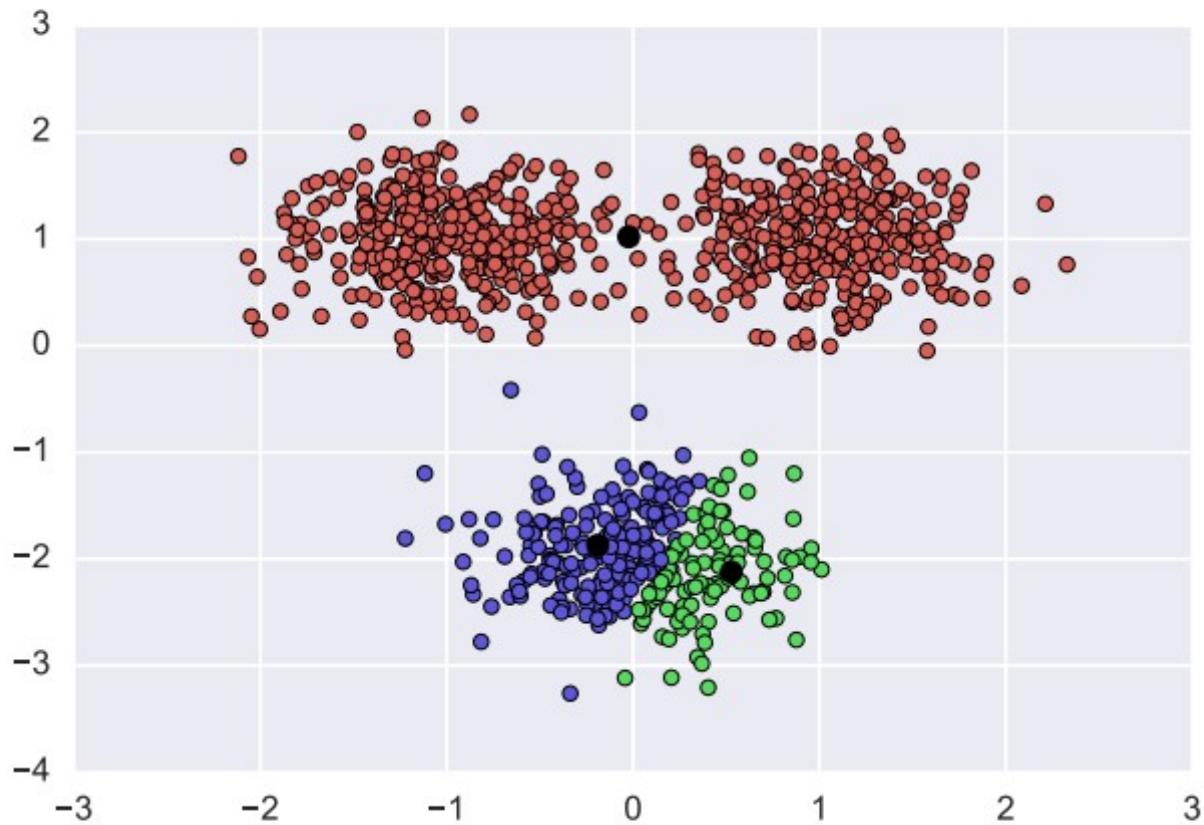
Since the k-means objective function has local optima, there is the chance that we convert to a less-than-ideal local optima

Especially for large/high-dimensional datasets, this is not hypothetical: k-means will usually converge to a different local optima depending on its starting point

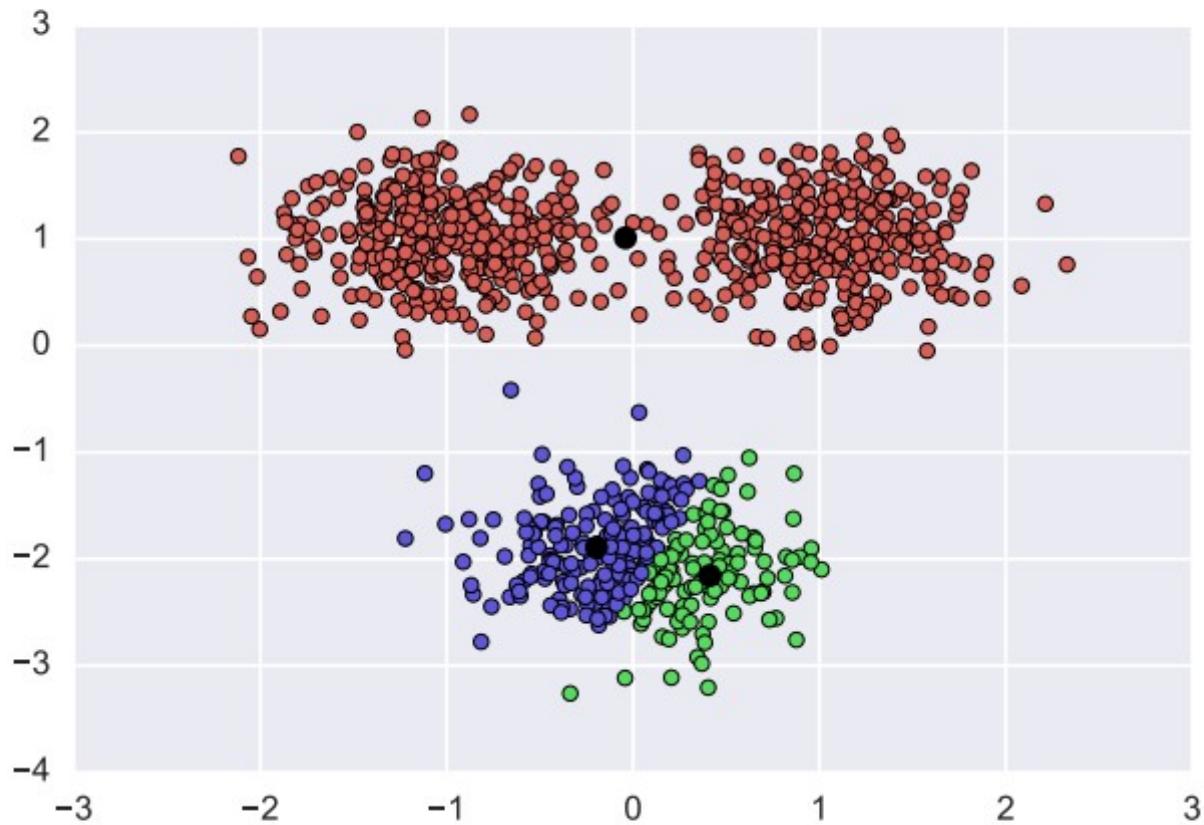
Convergence of k-means (bad)



Convergence of k-means (bad)



Convergence of k-means (bad)



Addressing poor clusters

Many approaches to address potential poor clustering: e.g. randomly initialize many times, take clustering with lowest loss

A common heuristic, k-means++: when initializing means, don't select $\mu^{(i)}$ randomly from all clusters, instead choose $\mu^{(i)}$ sequentially, sampled with probability proportion to the minimum squared distance to all other centroids

After these centers are initialized, run k-means as normal

K-means++

Given: Data set $(x^{(i)})_{i=1,\dots,m}$, # clusters k

Initialize:

$$\mu^{(1)} \leftarrow \text{Random}(x^{(1:m)})$$

For $j = 2, \dots, k$:

Select new cluster:

$$\mu^{(j)} \leftarrow \text{Random}(x^{(1:m)}, p^{(1:m)})$$

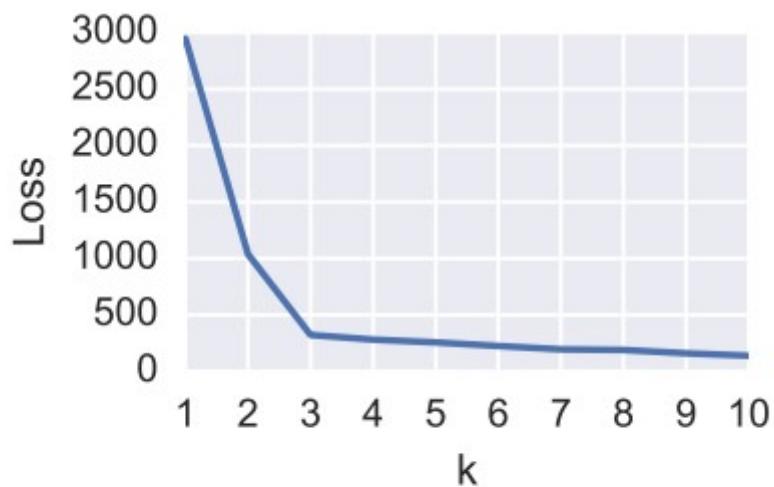
where probabilities $p^{(i)}$ given by

$$p^{(i)} \propto \min_{j' < j} \|\mu^{(j')} - x^{(i)}\|_2^2$$

How to select k?

There's no "right" way to select k (number of clusters): larger k virtually always will have lower loss than smaller k, even on a hold out set

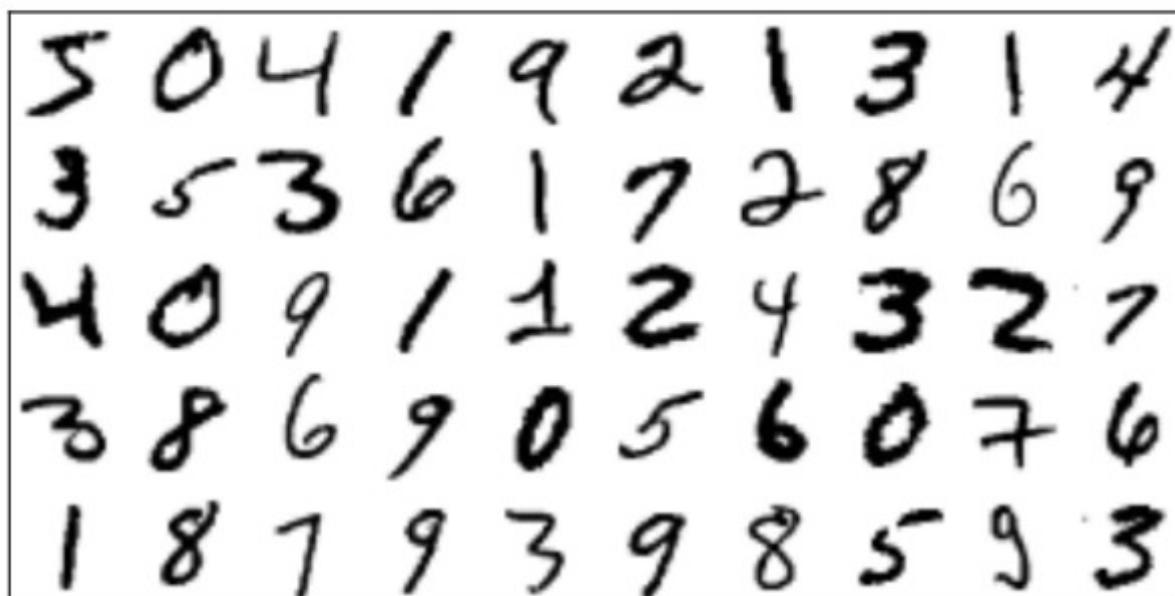
Instead, it's common to look at the loss function as a function of increasing k, and stop when things look "good" (lots of other heuristics, but they don't convincingly outperform this)



Example on real data

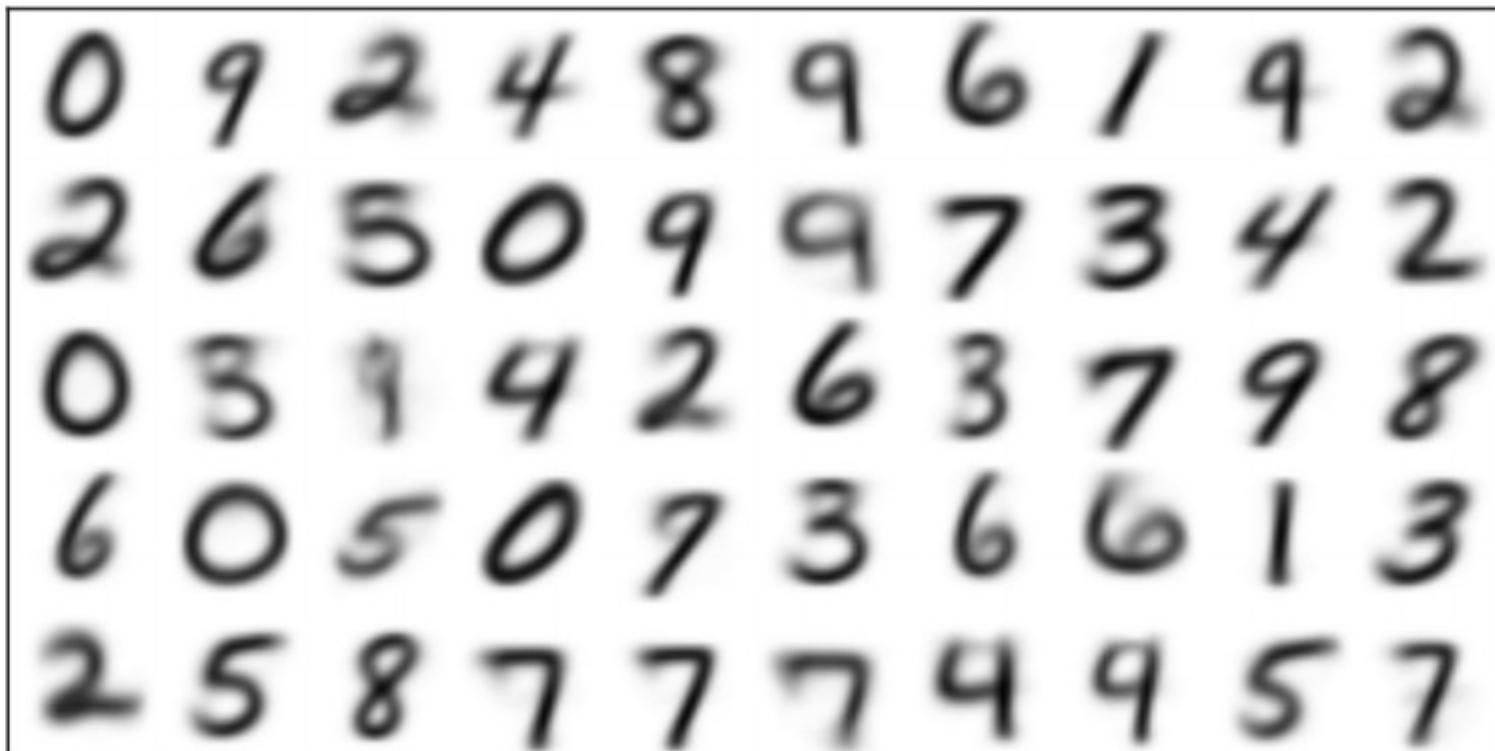
MNIST digit classification data set (used in question for 688 HW4)

60,000 images of digits, each 28x28

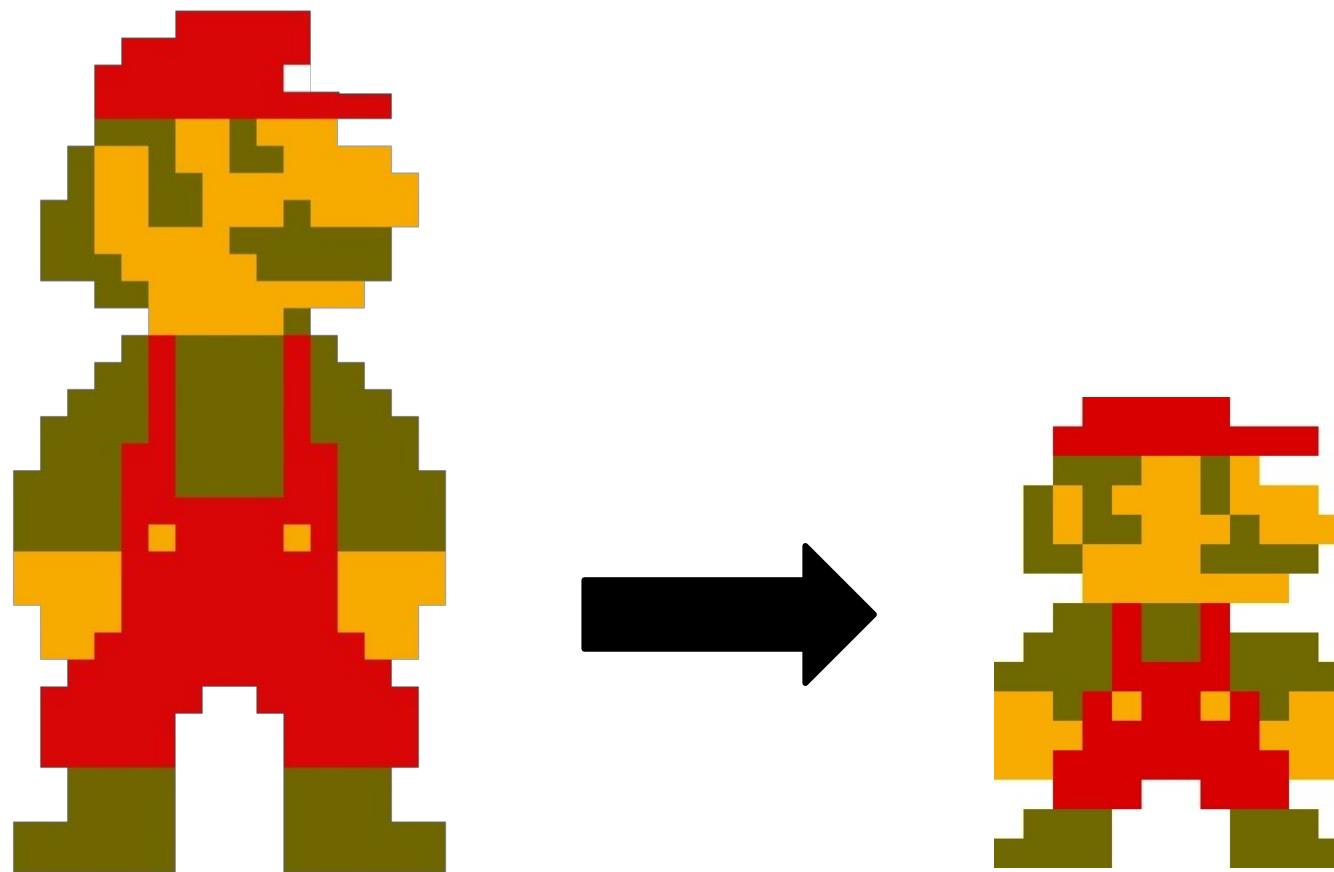


K-means run on MNIST

Means for k-means run with k=50 on MNIST data



DIMENSIONALITY REDUCTION



Thanks to: Zico Kolter

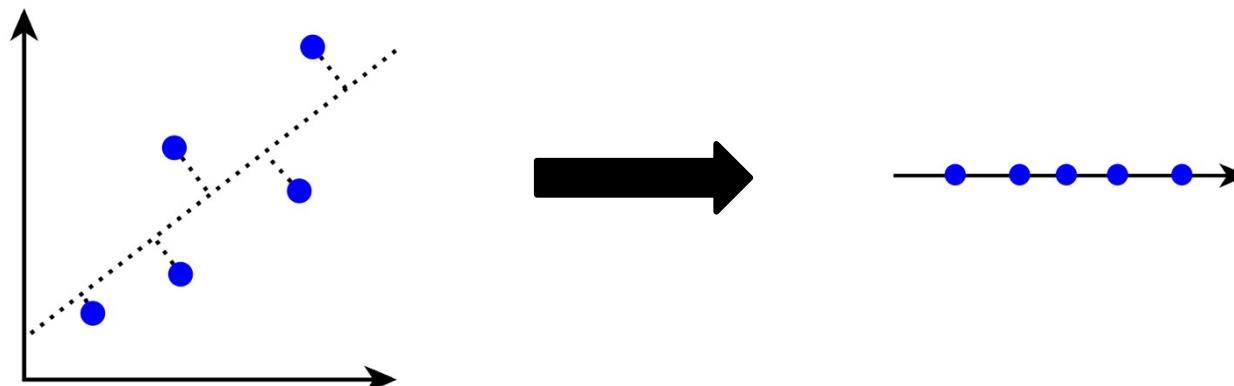
PRINCIPAL COMPONENT ANALYSIS (PCA)

So you've measured lots of features ...

- Overfitting, interpretability issues, visualization, computation

Can we combine raw features into new features that yield a simpler description of the same system?

Principal component analysis (PCA) does this by preserving the axis of major variation in the data:



PRINCIPAL COMPONENT ANALYSIS (PCA)

Assume: data is normalized ??????????

- Zero mean, unit (= 1) variance

Hypothesis function:

$$h_{\theta}(x) = UWx, \theta = \{U \in \mathbb{R}^{n \times k}, W \in \mathbb{R}^{k \times n}\}$$

- First multiply input by low rank matrix W (“compress” it), then map back into the initial space using U

Loss function: squared distance (like k-means)

$$\ell(h_{\theta}(x), x) = \|h_{\theta}(x) - x\|_2^2$$

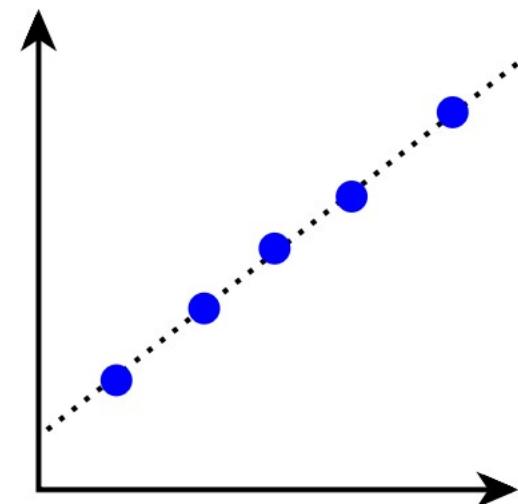
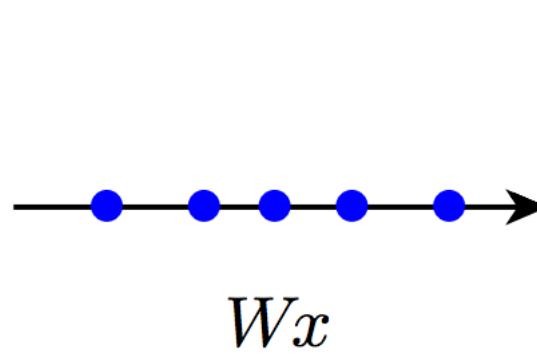
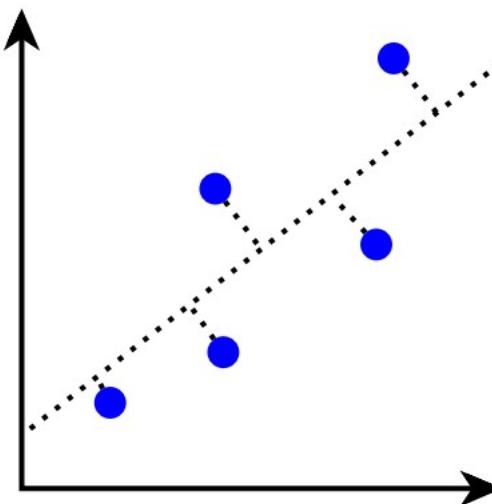
Optimization problem:

$$\underset{U,W}{\text{minimize}} \sum_{i=1}^m \|UWx^{(i)} - x^{(i)}\|_2^2$$

PRINCIPAL COMPONENT ANALYSIS (PCA)

Dimensionality reduction: main use of PCA for data science applications

If $h_{\theta}(x) = UWx$, then $Wx \in \mathbb{R}^k$ is a reduced (probably with some loss) representation of input features x



PRINCIPAL COMPONENT ANALYSIS (PCA)

CMSC422
MATH240

$$\underset{U, W}{\text{minimize}} \sum_{i=1}^m \|UWx^{(i)} - x^{(i)}\|_2^2$$

PCA optimization problem is non-convex ????????????

We can solve the problem exactly using the singular value decomposition (SVD):

- Factorize matrix $M = U \Sigma V^T$ (also used to approximate)

$m \times n$

\approx

$m \times r$

$r \times r$

$r \times n$

M

U

Σ

V^T

PCA IN PYTHON

Can roll your own PCA easily (assuming a call to SVD via SciPy or similar) ...

... or just use Scikit-Learn:

```
from sklearn.decomposition import PCA  
  
X=np.array([[-1,-1], [-2,-1], [-3,-2], [1,1], [2,1], [3,2]])  
  
# Fit PCA with 2 components (i.e., two final features)  
pca = PCA(n_components=2)  
pca.fit(X)  
print(pca.explained_variance_ratio_)
```

```
[ 0.99244... 0.00755...]
```



Looks like our data basically sit on a line

HOW TO USE PCA & FRIENDS IN PRACTICE

Unsupervised learning methods are useful for EDA

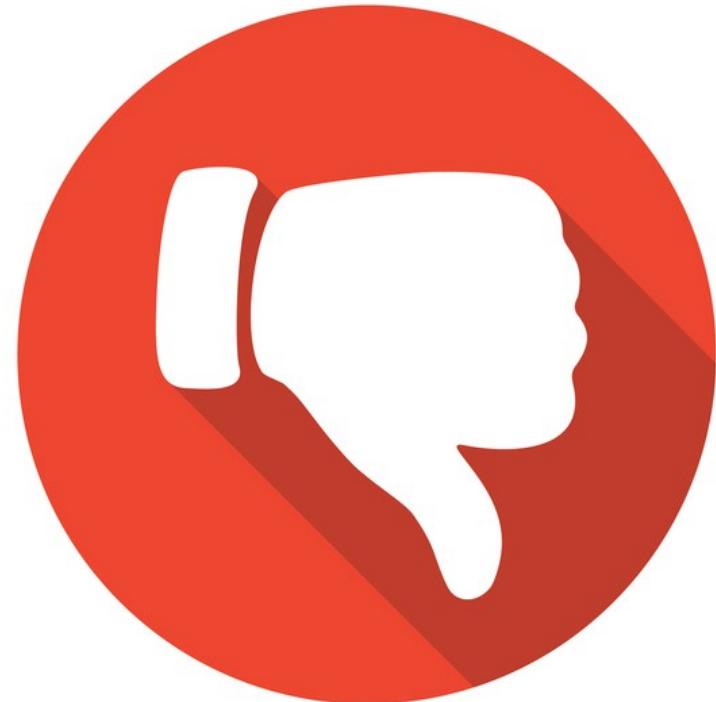
- Cluster or reduce to a few dimensions and visualize!

Also useful as data prep before supervised learning!

1. Run PCA, get W matrix
2. Transform $\tilde{x}^{(i)} = Wx^{(i)}$ – (reduce colinearity, dimension)
3. Train and test your favorite supervised classifier

Or use k-means to set up radial basis functions (RBFs):

4. Get k centers $\mu^{(1)}, \dots, \mu^{(k)}$
5. Create RBF features $\phi_j^{(i)} = \exp\left(-\frac{\|x^{(i)} - \mu^{(j)}\|_2^2}{2\sigma^2}\right)$



RECOMMENDER SYSTEMS & COLLABORATIVE FILTERING

NETFLIX PRIZE

Recommender systems: predict a user's rating of an item

	Twilight	Wall-E	Twilight II	TFotF
User 1	+1	-1	+1	?
User 2	+1	-1	?	?
	-1	+1	-1	+1

Netflix Prize: \$1MM to the first team that beats our in-house engine by 10%

- Happened after about three years
- Model was **never used** by Netflix for a variety of reasons
 - Out of date (DVDs vs streaming)
 - Too complicated / not interpretable

RECOMMENDER SYSTEMS

Recommender systems feel like:

- Supervised learning (we know the user watched some movies, so these are like labels)
- Unsupervised learning (we want to find latent structure, e.g., genres of movies)

They fall somewhere in between, in “Information Filtering” or Information Retrieval” ...

- ... but we can still just phrase the problem in terms of hypothesis classes, loss functions, and optimization problems

PREDICTION

Pure user information:

- Age
- Location
- Profession/Salary

Pure item information:

- Movie budget
- Main actors
- Is it a Netflix release?

User-item information:

- Which items are most similar to those I've watched before?
- Which users are most similar to me, and what did they watch?



COLLABORATIVE FILTERING

Collaborative filtering (CF): recommender systems that predict based only on the expressed preferences of other users for an item

$X =$

	i_1	i_2	i_3	i_4
u_1	1		3	
u_2		2	5	
u_3		3	5	
	4		4	

Rows are users

Cols are items

MATRIX VIEW

Goal: “fill in” the matrix

	i_1	i_2	i_3	i_4
u_1	1	?	?	3
u_2	?	2	5	?
u_3	?	3	?	5
	4	?	4	?

The matrix is **sparse**, but the empty cells are not (necessarily) zero!

APPROACHES TO CF

User-user:

- Find users who look like me – based on items that we've both rated
- Predict scores for my unrated items as average of those users

Item-item:

- Find similar items (based on scores from all users who have rated), predict scores for other users based off this

Matrix factorization:

- Find a low-rank decomposition of X that agrees (exactly, approximately) at the observed values

APPROACH #1: ITEM-BASED CF EX: INFER (USER 1, ITEM 3)

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	8	1	?	2	7
User 2	2	?	5	7	5
User 3	5	4	7	4	7
User 4	7	1	7	3	8
User 5	1	7	4	6	?
User 6	8	3	8	3	7

HOW TO CALCULATE SIMILARITY (ITEMS 3 AND 5)?

	Item 1	Item 2	Item 3	Item 4	Item 5
User 1	8	1	?	2	7
User 2	2	?	5	7	5
User 3	5	4	7	4	7
User 4	7	1	7	3	8
User 5	1	7	4	6	?
User 6	8	3	8	3	7

SIMILARITY BETWEEN ITEMS

Item 3	Item 4	Item 5
?	2	7
5	7	5
7	4	7
7	3	8
4	6	?
8	3	7

How should we calculate the similarity between two items (e.g., items 3 and 5)?

We've done this before in a different context!

SIMILARITY BETWEEN ITEMS

Item 3	Item 5
?	7
5	5
7	7
7	8
4	?
8	7

Only consider users (i.e., rows) who have rated both items (i.e., non-empty)

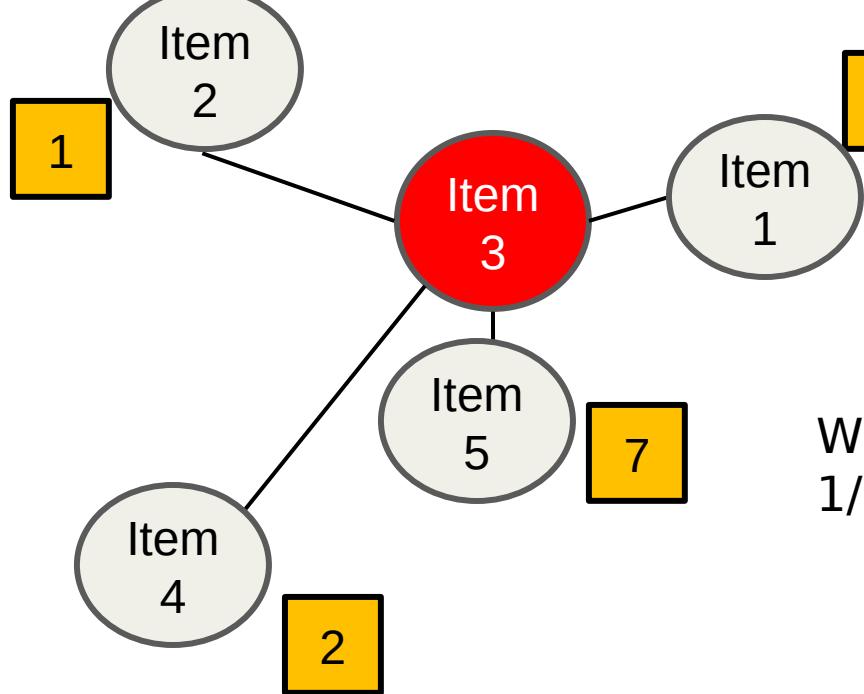
One approach: For each user:
Calculate difference in ratings
for the
two items
Take the average of this
difference

$$\text{similarity}(\text{item 3}, \text{item 5}) = \cosine((5, 7, 7, 8), (5, 7, 8))$$

over the users

$$\begin{aligned} &= \frac{(5*5 + 7*7 + 7*8 + 8*7)}{\sqrt{5^2 + 7^2 + 7^2 + 8^2}} \\ &= \frac{(25 + 49 + 56 + 56)}{\sqrt{25 + 49 + 56 + 56}} \end{aligned}$$

PREDICTION: CALCULATING RANKING $R(\text{USER1}, \text{ITEM3})$



$$r(\text{user}_1, \text{item}_3) = \alpha * \{ r(\text{user}_1, \text{item}_1) \text{sim}(\text{item}_1, \text{item}_3)$$

$$+ r(\text{user}_1, \text{item}_2) \text{sim}(\text{item}_2, \text{item}_3)$$

$$+ r(\text{user}_1, \text{item}_4) \text{sim}(\text{item}_4, \text{item}_3)$$

$$+ r(\text{user}_1, \text{item}_5) \text{sim}(\text{item}_5, \text{item}_3) \}$$

Where α is a normalization factor, which is $1/[\text{the sum of all sim}(\text{item}_i, \text{item}_3)]$.

APPROACH #2: CF VIA MATRIX FACTORIZATION

Want: all entries i, j of ratings matrix $X \in \mathbb{R}^{m \times n}$

Idea: approximate as $u_i^T v_j$

- User-specific weights $u_i \in \mathbb{R}^k$
- Item-specific weights $v_j \in \mathbb{R}^k$

Hypothesis function: ????????????

$$h_\theta(i, j) = u_i^T v_j, \quad \theta = \{u_{1:m}, v_{1:n}\}$$

Loss function: least squares on observed entries

$$\ell(h_\theta(i, j), X_{ij}) = (h_\theta(i, j) - X_{ij})^2$$

Optimization problem: (S is observed entries)

$$\underset{\theta}{\text{minimize}} \sum_{i, j \in S} \ell(h_\theta(i, j), X_{ij})$$

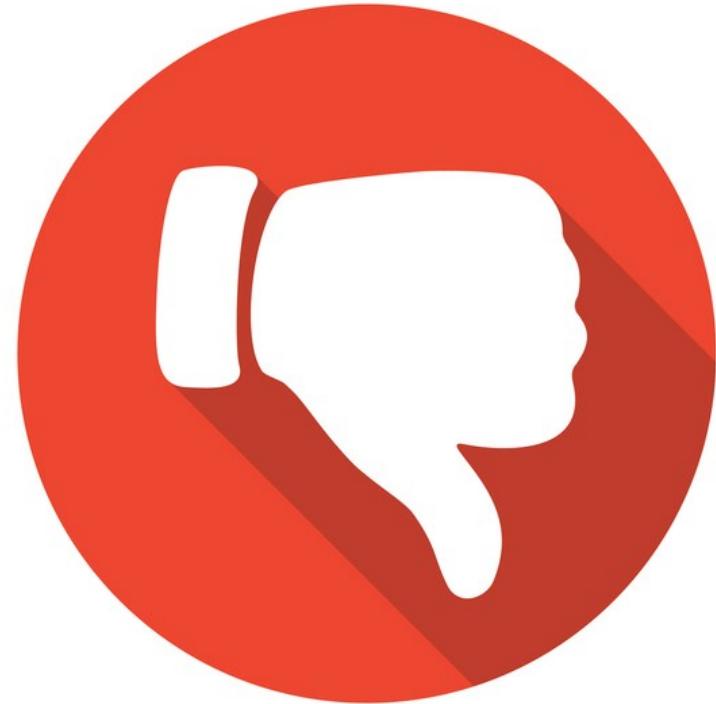
ISN'T THIS JUST PCA?

PCA also performs a factorization $X \approx UV$

- Or more precisely $X^T = UV = U(Wx)$

In PCA, all entries are observed (or imputed beforehand)

Simplifies the solution to PCA (which we can solve exactly via the SVD) versus CF via matrix factorization (which we cannot)



(SOME MORE)
RECOMMENDER SYSTEMS (ISH)

ASSOCIATION RULES

Last time: CF systems give predictions based on other users' scores of the same item

Complementary idea: Find rules that **associate** the presence of one set of items with that of another set of items

Customers who bought this item also bought



ThinkGeek Plush Unicorn Slippers, One Size, White
★★★★★ 395
\$7.77



Adult New Purple Unicorn Onesie Pajamas Kigurumi Cosplay Costumes Animal Outfit
★★★★★ 168
\$23.99 - \$28.99



EOS ~ Holiday 2015 Limited Edition Decorative Lip Balm Collection
★★★★★ 156
\$5.24 - \$22.99

FORMAT OF ASSOCIATION RULES

Typical Rule form:

- **Body \rightarrowtail Head**
- **Body and Head can be represented as sets of items (in transaction data) or as conjunction of predicates (in relational data)**
- **Support and Confidence**
 - Usually reported along with the rules
 - Metrics that indicate the strength of the item associations

Examples:

- {diaper, milk} \rightarrowtail {beer} [support: 0.5%, confidence: 78%]
- buys(x, "bread") \wedge buys(x, "eggs") \rightarrowtail buys(x, "milk") [sup: 0.6%, conf: 65%]
- major(x, "CS") \wedge takes(x, "DB") \rightarrowtail grade(x, "A") [1%, 75%]
- age(x, 30-45) \wedge income(x, 50K-75K) \rightarrowtail owns(x, SUV)
- age="30-45", income="50K-75K" \rightarrowtail car="SUV"

ASSOCIATION RULES: BASIC CONCEPTS

Let D be database of transactions

Transaction ID	Items
1000	A, B, C
2000	A, B
3000	A, D
4000	B, E, F

Let I be the set of items that appear in the database:

- e.g., $I = \{A, B, C, D, E, F\}$

Each transaction t is a subset of I

A rule is an implication among itemsets X and Y, of the form by $X \rightarrow Y$,
where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$

- e.g.: $\{B, C\} \rightarrow \{A\}$

ASSOCIATION RULES: BASIC CONCEPTS

Itemset

- A set of one or more items
 - E.g.: {Milk, Bread, Diaper}
- k-itemset
 - An itemset that contains k items

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Support count (σ)

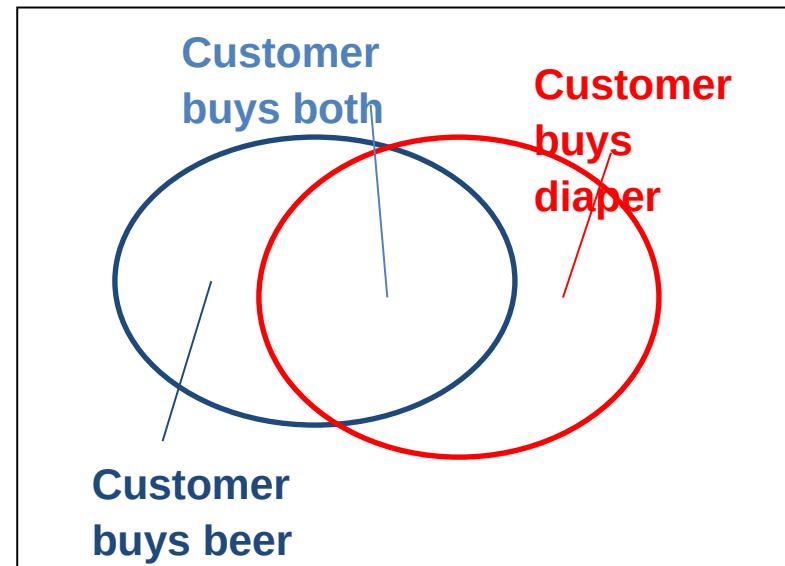
- Frequency of occurrence of an itemset (number of transactions in which it appears)
- E.g. $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$

Support

- Fraction of the transactions in which an itemset appears
- E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 2/5$

Frequent Itemset

- An itemset whose support is greater than or equal to a $minsup$ threshold



ASSOCIATION RULES: BASIC CONCEPTS

Association Rule

- $X \rightarrow Y$, where X and Y are non-overlapping itemsets
- $\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$

Rule Evaluation Metrics

- **Support (s)**
 - Fraction of transactions that contain both X and Y
 - i.e., support of the itemset $X \cup Y$
- **Confidence (c)**
 - Measures how often items in Y appear in transactions that contain X

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$\{\text{Milk, Diaper}\} \rightarrow \text{Beer}$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|D|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

ASSOCIATION RULES: INTERESTINGNESS

Another interpretation of support and confidence for $X \rightarrow Y$

- **Support** is the **probability** that a transaction contains $\{X \cup Y\}$ or $\Pr(X \wedge Y)$

$$\text{support}(X \rightarrow Y) = \text{support}(X \cup Y) = \sigma(X \cup Y) / |D|$$

- **Confidence** is the **conditional probability** that a transaction will contain Y given that it contains X or $\Pr(Y | X)$

$$\begin{aligned}\text{confidence}(X \rightarrow Y) &= \sigma(X \cup Y) / \sigma(X) \\ &= \text{support}(X \cup Y) / \text{support}(X)\end{aligned}$$

ASSOCIATION RULES: INTERESTINGNESS

Other considerations of how interesting a rule is:

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)}$$

If lift is equal to 1 ??????????

- Body X and Head Y are independent

If lift is greater than 1 ??????????

- Body X and Head Y are in some sense dependent

Conviction measures frequency of X and Y occurring together, vs. how frequently X occurs but not Y

Many others ...

ASSOCIATION RULES IN PRACTICE

Orange3 is a {GUI, Python API, ...} that:

- Enumerates frequent itemsets
- Performs association rule mining
- (Wrapper calls to, shared functionality with, Scikit-Learn)

```
conda install -c ales-erjavec orange3
```

More information:

<https://blog.biolab.si/2016/04/25/association-rules-in-orange/>

In general:

- Can be useful for interpretable, fast data mining
- Typically doesn't consider order, scalability issues ...