*April 25, 2018*

# CS 361: Probability & Statistics

Clustering & vector quantization

# k-means clustering

# Another approach

A completely different approach to clustering is given results from the following observations

If we had to produce a clustering of a set of points into $k$ clusters, and we knew where the center of each cluster was, it would be easy to assign points to clusters. We would just assign a point to the cluster corresponding to the cluster center it is nearest to

If instead we knew which points belonged to which clusters, it is quite easy to calculate the center of the cluster

# An algorithm

Taking these two observations into account a natural approach would be to try and minimize the following function by finding an optimal allocation of clusters

$$\Phi(\delta, \mathbf{c}) = \sum_{i,j} \delta_{i,j} \left[ (\mathbf{x}_i - \mathbf{c}_j)^T (\mathbf{x}_i - \mathbf{c}_j) \right]$$

where

$$\delta_{i,j} = \begin{cases} 1 \text{ if } \mathbf{x}_i \text{ belongs to cluster } j \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$

where c_j is the cluster center of the j-th cluster and x_i is the i-th data point

Unfortunately there are an exponential number of different potential assignments of points to clusters, so we will need a different approach than to directly minimize this function

# k-means

The method of **k-means clustering** takes our two observations into account in the following way.

First we assume we know that there are $k$ clusters

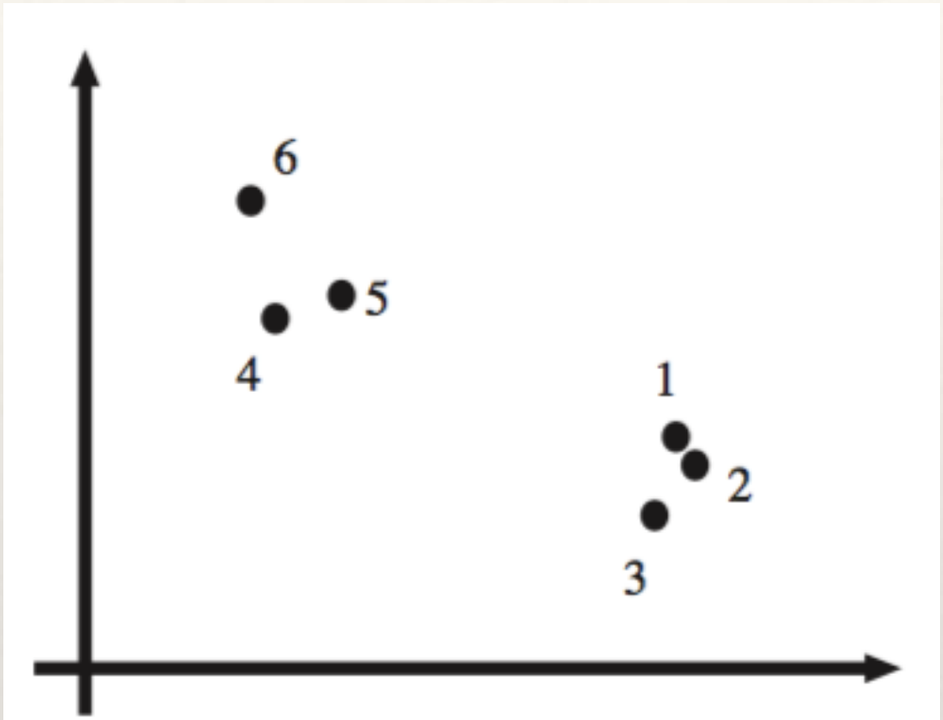
We initialize by picking $k$ random points in our dataset and calling them cluster centers

Then, until our clustering changes very little, we run the following loop

(Re-)Assign each point to the cluster with the most nearby center

Calculate the center of all of the clusters

# Example

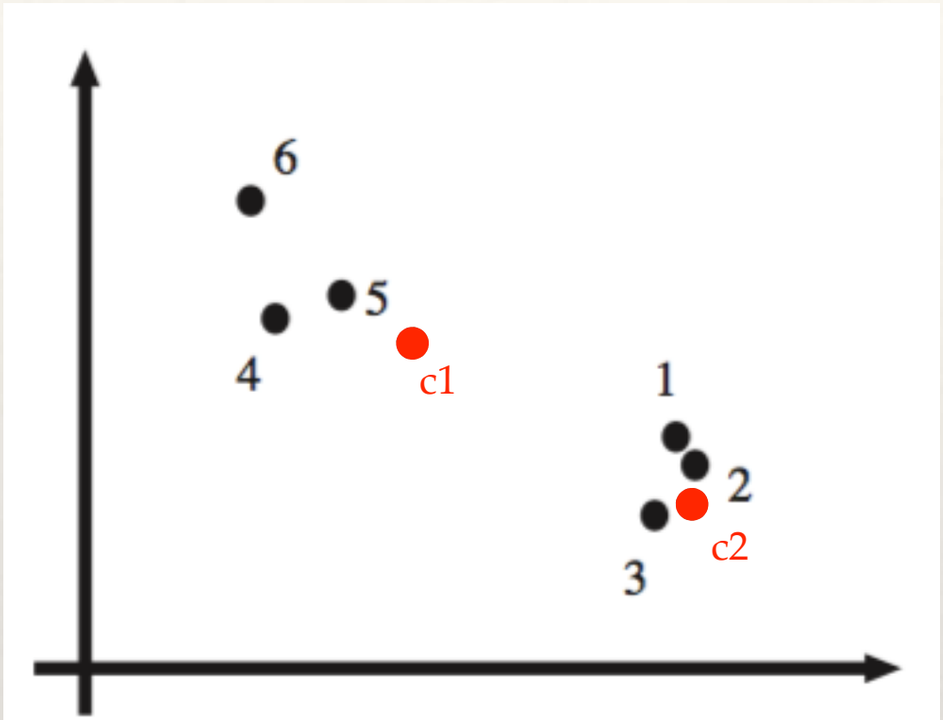



Let's say that k=2

And we randomly choose point 1 to be the center of cluster 1 and point 2 to be the center of cluster 2

Then we will have
c1 = {1,4,5,6} and c2={2,3}

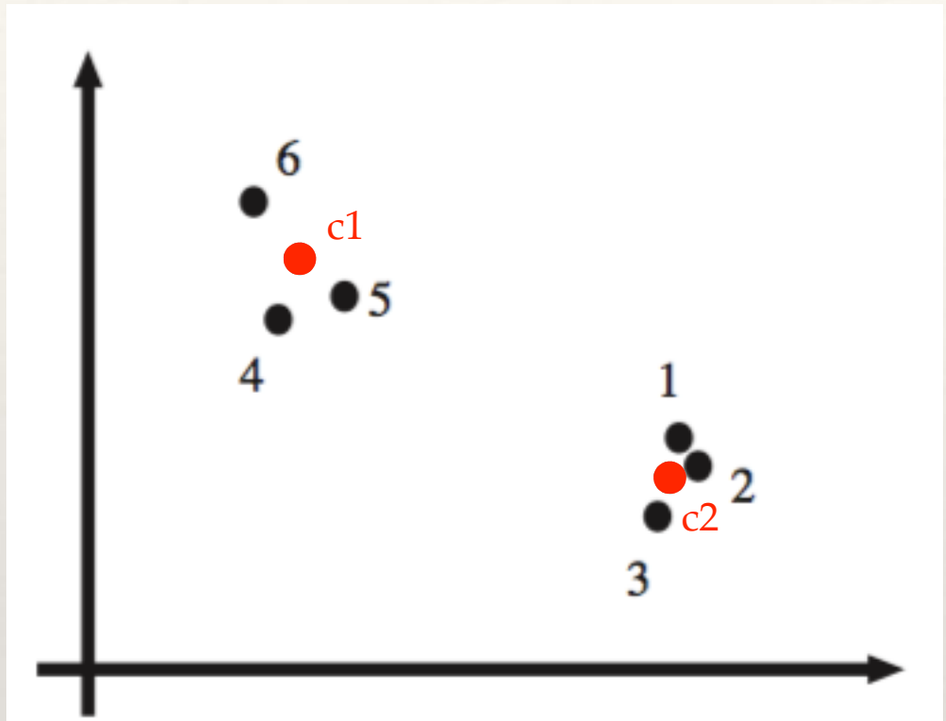
# Example




With c1 = {1,4,5,6} and c2={2,3}, we re-calculate the cluster centers as just the mean of all the points in the cluster getting

We then re-assign points to their nearest cluster centers getting
c1={4,5,6} and c2={1,2,3}

# Example




With c1={4,5,6} and c2={1,2,3} we recalculate cluster centers getting

We then re-assign points to the nearest cluster centers getting
c1={4,5,6} and c2={1,2,3}

This represents no change from the last iteration, so we stop

# A couple of caveats

We need to figure out how to choose $k$

Doesn't this algorithm potentially produce a different clustering depending on which start points we use?

# Choosing k

One approach is to try different values for *k* and calculate the value of the cost function for each

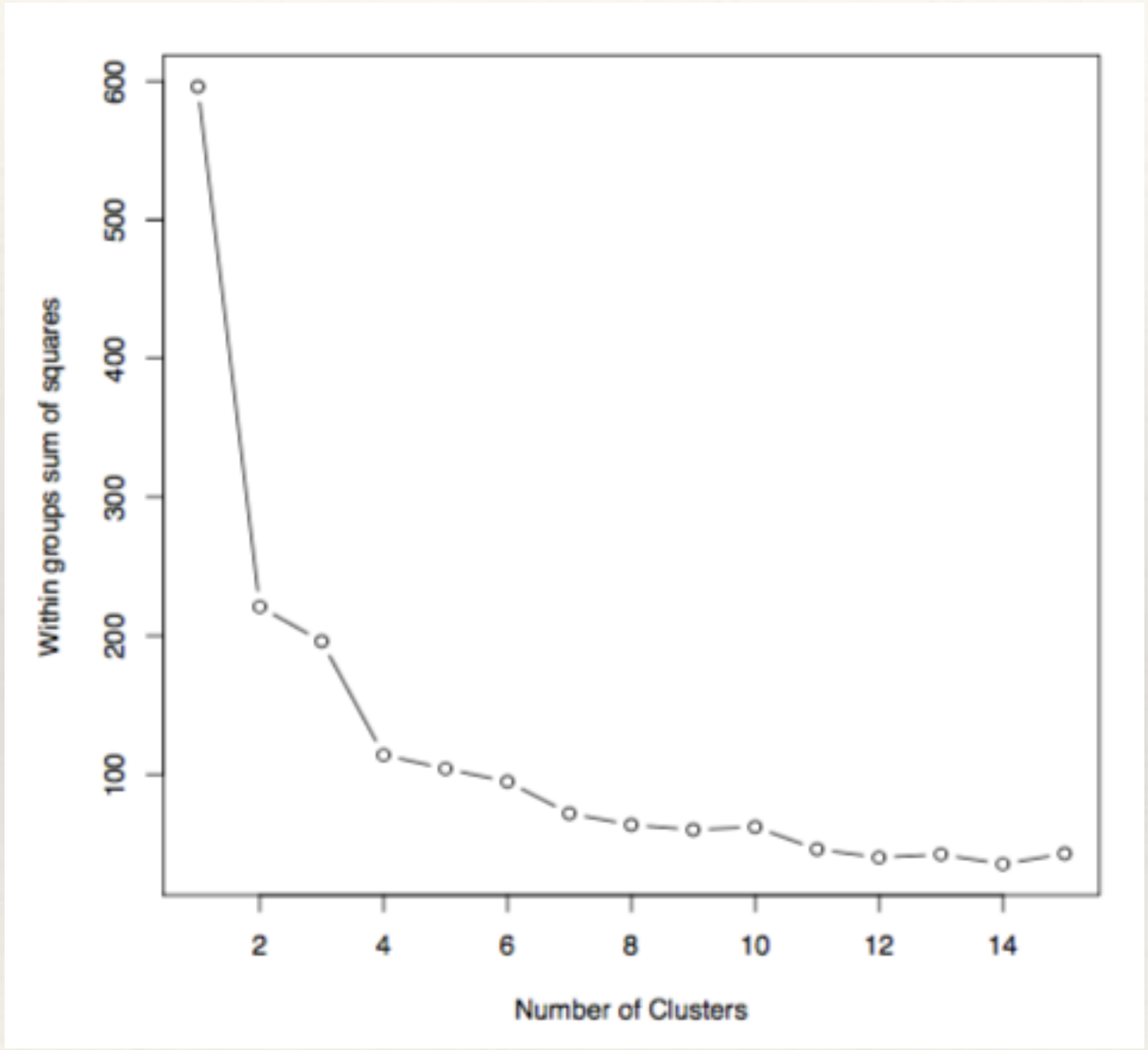
$$\Phi(\delta, \mathbf{c}) = \sum_{i,j} \delta_{i,j} \left[ (\mathbf{x}_i - \mathbf{c}_j)^T (\mathbf{x}_i - \mathbf{c}_j) \right]$$

where

$$\delta_{i,j} = \begin{cases} 1 \text{ if } \mathbf{x}_i \text{ belongs to cluster } j \\ 0 \quad \text{otherwise} \end{cases}$$

As *k* increases this function decreases since a larger *k* means there are more centers — so a point should be able to find a center that is close to it. So we really want to find a "knee" on the curve and stop there

# Choosing k




A plot for the iris dataset, which we know to have 3 classes

# Comments on k-means

If you use k-means enough you will notice that it almost always produces some very spread out clusters or some single element clusters. While most clusters are fairly blobby, there are usually one or more bad clusters

Since each data item must belong to a cluster, this isn't unexpected. Data points that are far from all the others will either wind up on their own or they will be assigned to a cluster and drag its center far from where it would otherwise be

One possible solution is to have a junk cluster. We set some threshold and if during the step where we assign points to clusters, the distance between a point and its nearest cluster center is greater than this threshold, we assign it to the junk cluster

If we do this, it's important to make sure that points can leave the junk cluster if a cluster center is close enough to them in a future iteration

# Hierarchical k-means

If we have a very large dataset, we can speed things up by sampling the dataset, clustering only the sampled data, and then assigning each point to the resulting clusters

If we then sub-cluster each individual cluster, looking at all of its data, we have a two-level tree of clusters that we can assign new points to. We could repeat this process many times, sampling at each internal node and only using all the data at the leaves in this tree of clusters

# Vector quantization

# Classification and vectors

In this chapter and the classification chapter we gave some algorithms for doing machine learning on vectors of features

Perhaps the biggest part of machine learning in practice is figuring out *which* features to use

Suppose I was developing a classifier to decide which users of Facebook to target with an ad. The input to this classifier are the nodes of a graph, not vectors, so somehow I need to be able to look at a node in this graph and generate a vector of features

You could probably think of 10 features of a Facebook user that might work well for this task, and the point is that doing this kind of thinking often has a surprisingly high ROI for the ultimate accuracy of your classifier

# Patterns

Images, videos, sounds, time series data, and accelerometer signals can also be the input of classification algorithms. We will be looking at a method for getting good features out of a variety of signals like these. An additional wrinkle with this kind of data is that each data point might be of a **different size** than the others

The key to this approach is that it exploits repetition. And most signals—like images or human speech sounds—can be thought of as a composition of a relatively small vocabulary of patterns

# Patterns

An example is human speech. When we learn to read or learn a new language we spend a large amount of time “sounding things out” — we learn the phonemes (things like k- and th- ) of the language and learn how to combine them in order to produce fluent human speech

This small vocabulary of phonemes is all we need to pronounce any given utterance

We can re-construct a given signal then, like a recording of some speech, as long as we know which patterns we should be using and how they should be combined to form the recording

# Patterns

We will develop an approach to look at a set of data and **extract these patterns** which we can then use to express any given data item as a feature vector—in particular a histogram of the counts of how many times each pattern appears

For the examples we will work on, this is all we will keep. We won't even store the order that the patterns appear in, just how many times each pattern appeared in a given item. This is sometimes enough to do classification on things like images and accelerometer data

# Vector quantization

We do the following to extract the pattern vocabulary. First, take the input and break it up into fixed-size sub signals. We might take an image and break it up into a bunch of 10x10 pixel sub images, for example

These sub signals could be overlapping or not. In the homework it probably isn't necessary for them to be overlapping

We do this for every image in our dataset, getting a large collection of subsamples

Then we just cluster this large collection of subsamples

The centers of these resulting clusters become our patterns. We give each cluster an integer index [1, …, $k$]

# Vector quantization

Now for any data item, in order to get a feature representation for it. We break it up into sub signals and for each sub signal we see which of the cluster centers it is closest to

A given data item then will become a histogram. Each of its sub signals contributes to the count of one of the $k$ patterns in the vocabulary, and we represent the item by this histogram of counts of the number of patterns in the item

# Example

Dataset of images of beaches




We slice each image up into 10x10 pixel sub-images

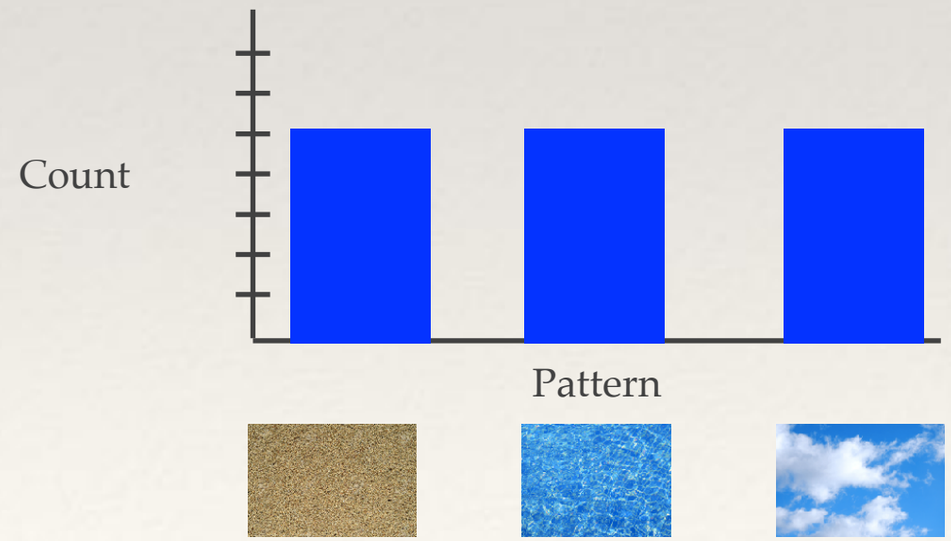
Then, looking at the set of all such 10x10 pixel sub-images we construct a **pattern vocabulary** of patterns that approximate many of the sub-images well



# Example

Now we can turn arbitrary images, even ones of differing sizes, into a fixed-length feature vector. To featurize an image we dice it up into 10x10 pixel sub images

And then for each sub-image, we see which of the members of the pattern vocabulary it is closest to, keeping a histogram of counts




query_image = [5,5,5]

# Example: the final

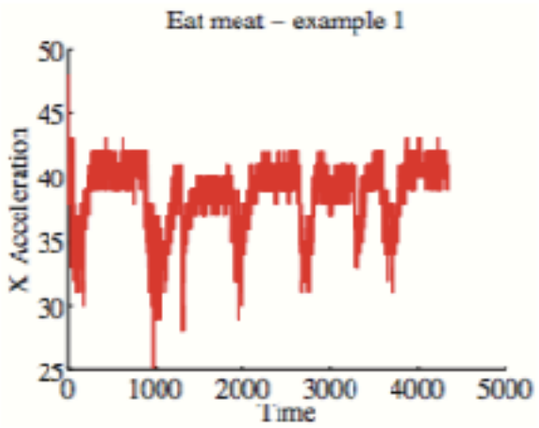
The textbook has a link to some accelerometer data, presumably from something like a FitBit

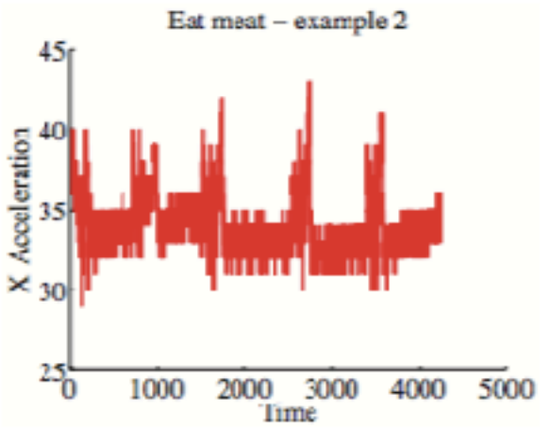
Each datafile corresponds to an instance of some activity like brushing your teeth and contains a long list of accelerations in the x, y, and z directions—32 measurements per second. The data files all have different lengths

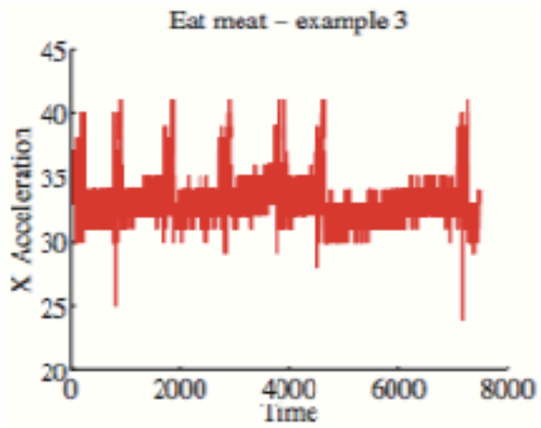
If we wanted to learn a classifier that could look at one of these files and output what activity is happening during the time period of interest, vector quantization is a good candidate for generating our features

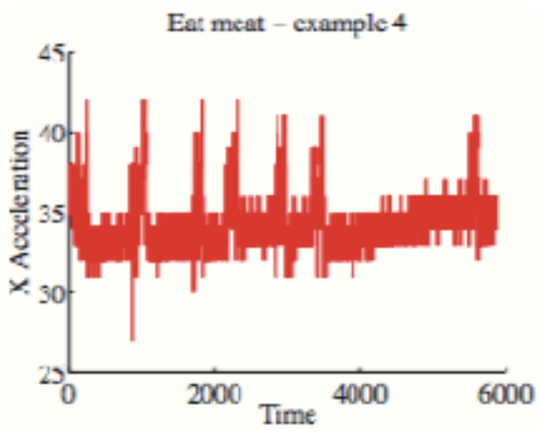
We wouldn't just input the raw data into a classifier because the data items aren't even of the same length

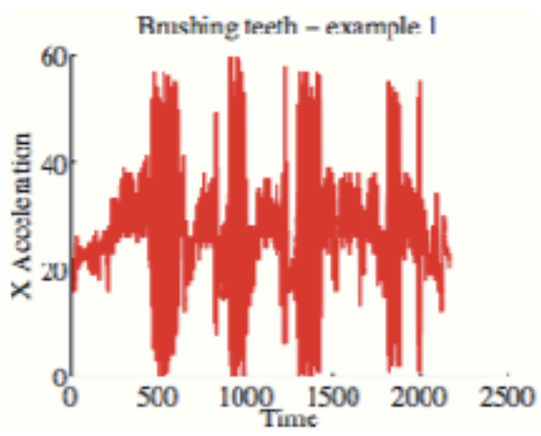
# Example

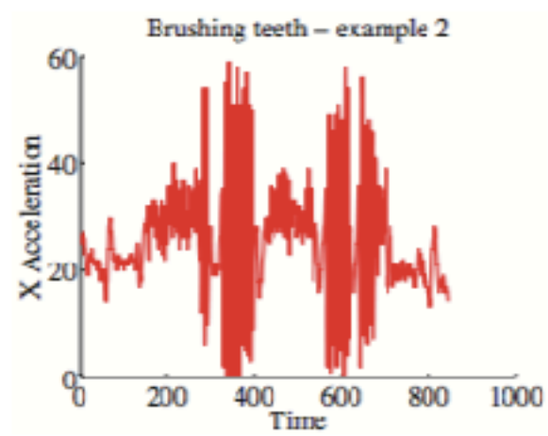

Eat meat – example 1
X Acceleration
Time

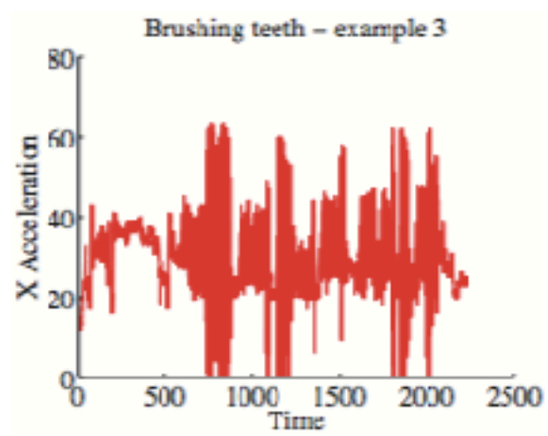

Eat meat – example 2
X Acceleration
Time

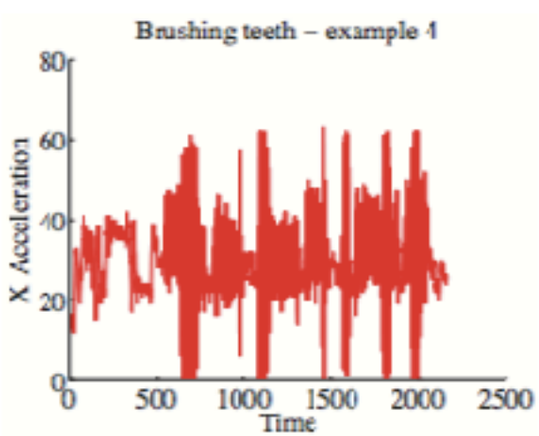

Eat meat – example 3
X Acceleration
Time


Eat meat – example 4
X Acceleration
Time


Brushing teeth – example 1
X Acceleration
Time


Brushing teeth – example 2
X Acceleration
Time


Brushing teeth – example 3
X Acceleration
Time

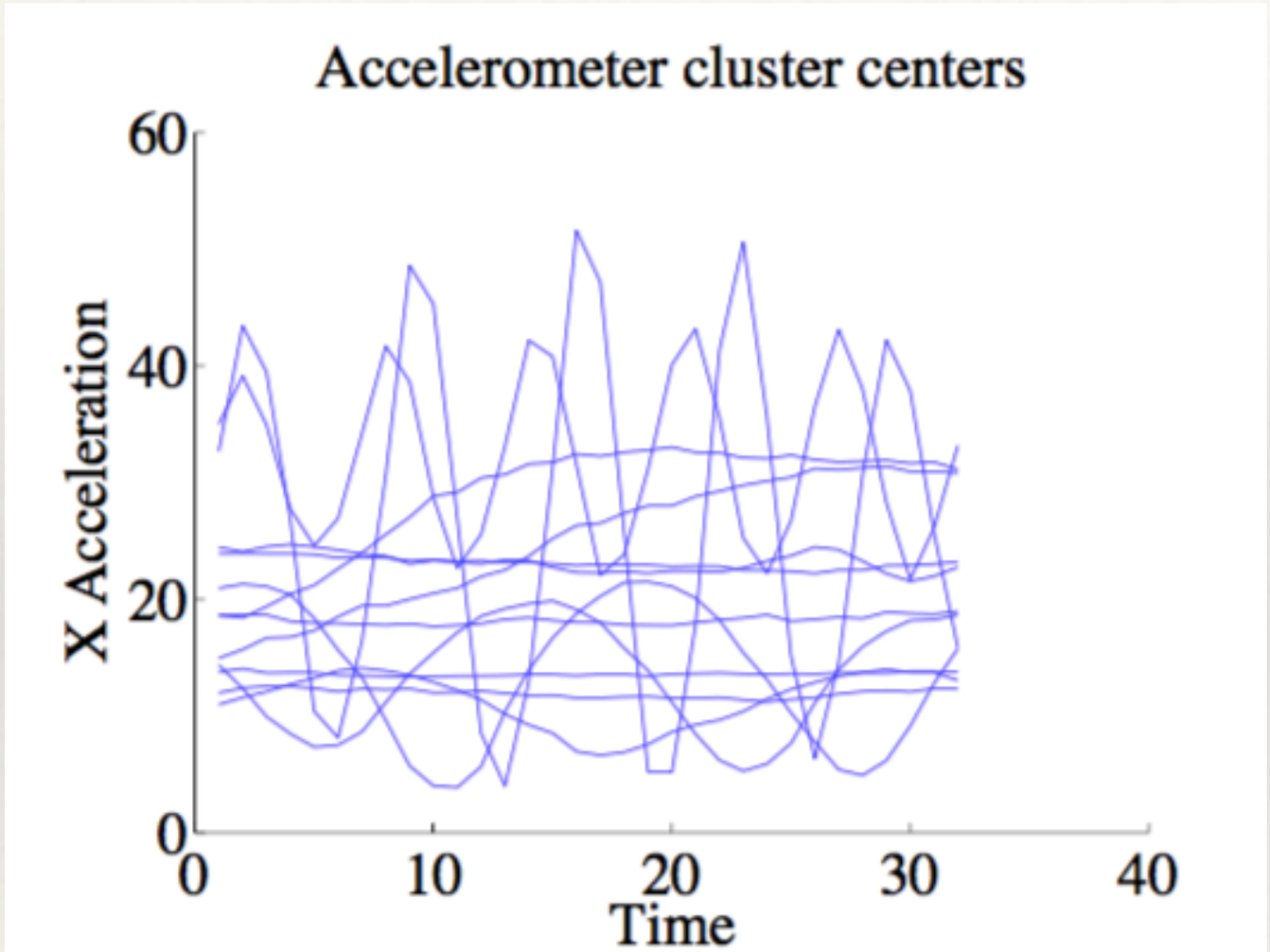

Brushing teeth – example 4
X Acceleration
Time

# Example

Let's take each data file and chop it up into non-overlapping sub signals of 32 measurements each (i.e. 1 second intervals)

We can then cluster the set of all sub signals. We get a set of clusters. Each cluster center will then be a set of 32 measurements, which we can look at

# Example

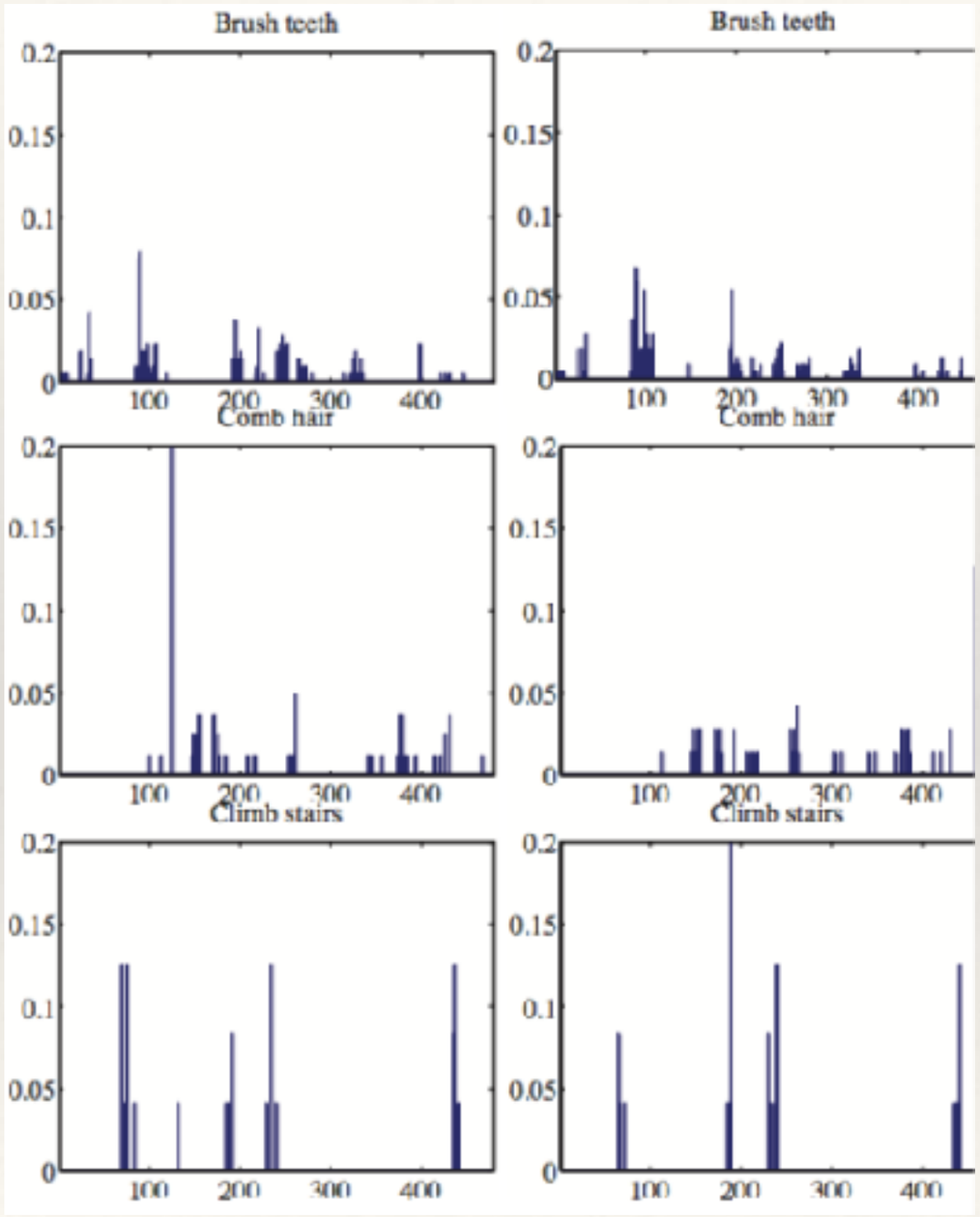



Some of the cluster centers

# Example

Keeping these cluster centers around and giving each cluster an integer index, we can now take any given activity file and break it up into sub signals, align each sub signal to the closest cluster center, and compute a histogram for that activity

# Example




Here we used hierarchical k-means, take a smaller sample of the data and clustering it into 40 clusters and then for each cluster, clustering the data in that cluster into 12 clusters, giving 480 total clusters/patterns

Our histograms have been normalized to give the relative frequency of each pattern rather than the raw count