K-means clustering and vector quantization (`scipy.cluster.vq`)

K-means Clustering and Vector Quantization Module

Provides routines for k-means clustering, generating code books from k-means models, and quantizing vectors by comparing them book.

The k-means algorithm takes as input the number of clusters to generate, k, and a set of observation vectors to cluster. It returns a each of the k clusters. An observation vector is classified with the cluster number or centroid index of the centroid closest to it.

A vector v belongs to cluster i if it is closer to centroid i than any other centroids. If v belongs to i, we say centroid i is the dominati variants of k-means try to minimize distortion, which is defined as the sum of the distances between each observation vector and Each step of the k-means algorithm refines the choices of centroids to reduce distortion. The change in distortion is often used as a the change is lower than a threshold, the k-means algorithm is not making sufficient progress and terminates.

Since vector quantization is a natural application for k-means, information theory terminology is often used. The centroid index or clu to as a "code" and the table mapping codes to centroids and vice versa is often referred as a "code book". The result of k-means, used to quantize vectors. Quantization aims to find an encoding of vectors that reduces the expected distortion.

For example, suppose we wish to compress a 24-bit color image (each pixel is represented by one byte for red, one for blue, and one it over the web. By using a smaller 8-bit encoding, we can reduce the amount of data by two thirds. Ideally, the colors for each encoding values should be chosen to minimize distortion of the color. Running k-means with k=256 generates a code book of 25 possible 8-bit sequences. Instead of sending a 3-byte value for each pixel, the 8-bit centroid index (or code word) of the dominatin The code book is also sent over the wire so each 8-bit code can be translated back to a 24-bit pixel value representation. If the im ocean, we would expect many 24-bit blues to be represented by 8-bit codes. If it was an image of a human face, more flesh tone co in the code book.

All routines expect obs to be a M by N array where the rows are the observation vectors. The codebook is a k by N array where the code word i. The observation vectors and centroids have the same feature dimension.

whiten(obs) –
: Normalize a group of observations so each feature has unit variance.

vq(obs,code_book) –
: Calculate code book membership of a set of observation vectors.

kmeans(obs,k_or_guess,iter=20,thresh=1e-5) –
: Clusters a set of observation vectors. Learns centroids with the k-means algorithm, trying to minimize distortion. A code book used to quantize vectors.

kmeans2 –
: A different implementation of k-means with more methods for initializing centroids. Uses maximum number of iterations as threshold as its stopping criterion.

`scipy.cluster.vq.`**`whiten`**(*obs*)

Normalize a group of observations on a per feature basis.

Before running k-means, it is beneficial to rescale each feature dimension of the observation set with whitening. Each feature deviation across all observations to give it unit variance.

Parameters: obs : ndarray

Each row of the array is an observation. The columns are the features seen during each observation.

```
      #   f0    f1    f2
obs = [[  1.,   1.,   1.],  #o0
       [  2.,   2.,   2.],  #o1
       [  3.,   3.,   3.],  #o2
       [  4.,   4.,   4.]]) #o3
```

XXX perhaps should have an axis variable here.

Returns: result : ndarray

Contains the values in obs scaled by the standard devation of each column.

Examples

```
>>> from numpy import array
>>> from scipy.cluster.vq import whiten
>>> features  = array([[  1.9,2.3,1.7],
...                    [  1.5,2.5,2.2],
...                    [  0.8,0.6,1.7,]])
>>> whiten(features)
array([[ 3.41250074,  2.20300046,  5.88897275],
       [ 2.69407953,  2.39456571,  7.62102355],
       [ 1.43684242,  0.57469577,  5.88897275]])
```

`scipy.cluster.vq.`**`vq`**(*obs*, *code_book*)

Vector Quantization: assign codes from a code book to observations.

Assigns a code from a code book to each observation. Each observation vector in the M by N obs array is compared with the c and assigned the code of the closest centroid.

The features in obs should have unit variance, which can be acheived by passing them through the whiten function. The code boo k-means algorithm or a different encoding algorithm.

Parameters: obs : ndarray

Each row of the NxM array is an observation. The columns are the "features" seen during each observatio whitened first using the whiten function or something equivalent.

code_book : ndarray.

The code book is usually generated using the k-means algorithm. Each row of the array holds a different are the features of the code.

```
             #   f0    f1    f2   f3
code_book = [[  1.,   2.,   3.,   4.],  #c0
             [  1.,   2.,   3.,   4.],  #c1
             [  1.,   2.,   3.,   4.]]) #c2
```

Returns:

code : ndarray
: A length N array holding the code book index for each observation.

dist : ndarray
: The distortion (distance) between the observation and its nearest code.

Notes

This currently forces 32-bit math precision for speed. Anyone know of a situation where this undermines the accuracy of the algor

Examples

```
>>> from numpy import array
>>> from scipy.cluster.vq import vq
>>> code_book = array([[1.,1.,1.],
...                    [2.,2.,2.]])
>>> features  = array([[  1.9,2.3,1.7],
...                    [  1.5,2.5,2.2],
...                    [  0.8,0.6,1.7]])
>>> vq(features,code_book)
(array([1, 1, 0],'i'), array([ 0.43588989,  0.73484692,  0.83066239]))
```

scipy.cluster.vq.**kmeans**(*obs*, *k_or_guess*, *iter=20*, *thresh=1.0000000000000001e-05*)

Performs k-means on a set of observation vectors forming k

clusters. This yields a code book mapping centroids to codes and vice versa. The k-means algorithm adjusts the centroid cannot be made, i.e. the change in distortion since the last iteration is less than some threshold.

Parameters:

obs : ndarray
: Each row of the M by N array is an observation vector. The columns are the features seen during each ob must be whitened first with the whiten function.

k_or_guess : int or ndarray
: The number of centroids to generate. A code is assigned to each centroid, which is also the row index code_book matrix generated.
: The initial k centroids are chosen by randomly selecting observations from the observation matrix. Alterna array specifies the initial k centroids.

iter : int
: The number of times to run k-means, returning the codebook with the lowest distortion. This argum centroids are specified with an array for the k_or_guess paramter. This parameter does not represent the the k-means algorithm.

thresh : float
: Terminates the k-means algorithm if the change in distortion since the last k-means iteration is less than th

Returns:

codebook : ndarray
: A k by N array of k centroids. The i'th centroid codebook[i] is represented with the code i. The centroid represent the lowest distortion seen, not necessarily the globally minimal distortion.

distortion : float
: The distortion between the observations passed and the centroids generated.

Seealso:

- kmeans2: a different implementation of k-means clustering with more methods for generating initial centro distortion change threshold as a stopping criterion.
- whiten: must be called prior to passing an observation matrix to kmeans.

Examples

```
>>> from numpy import array
>>> from scipy.cluster.vq import vq, kmeans, whiten
>>> features  = array([[ 1.9,2.3],
...                    [ 1.5,2.5],
...                    [ 0.8,0.6],
...                    [ 0.4,1.8],
...                    [ 0.1,0.1],
...                    [ 0.2,1.8],
...                    [ 2.0,0.5],
...                    [ 0.3,1.5],
...                    [ 1.0,1.0]])
>>> whitened = whiten(features)
>>> book = array((whitened[0],whitened[2]))
>>> kmeans(whitened,book)
(array([[ 2.3110306 ,  2.86287398],
       [ 0.93218041,  1.24398691]]), 0.85684700941625547)
```

```
>>> from numpy import random
>>> random.seed((1000,2000))
>>> codes = 3
>>> kmeans(whitened,codes)
(array([[ 2.3110306 ,  2.86287398],
       [ 1.32544402,  0.65607529],
       [ 0.40782893,  2.02786907]]), 0.5196582527686241)
```

scipy.cluster.vq.**kmeans2**(*data*, *k*, *iter=10*, *thresh=1.0000000000000001e-05*, *minit='random'*, *missing='warn'*)

Classify a set of observations into k clusters using the k-means

algorithm.

The algorithm attempts to minimize the Euclidian distance between observations and centroids. Several initialization methods are

Parameters:

data : ndarray
: A M by N array of M observations in N dimensions or a length M array of M one-dimensional observations.

k : int or ndarray
: The number of clusters to form as well as the number of centroids to generate. If minit initialization st ndarray is given instead, it is interpreted as initial cluster to use instead.

iter : int
: Number of iterations of the k-means algrithm to run. Note that this differs in meaning from the iters pa function.

thresh : float
: (not used yet).

minit : string
    Method for initialization. Available methods are ‘random’, ‘points’, ‘uniform’, and ‘matrix’:
    ‘random’: generate k centroids from a Gaussian with mean and variance estimated from the data.
    ‘points’: choose k observations (rows) at random from data for the initial centroids.
    ‘uniform’: generate k observations from the data from a uniform distribution defined by the data set (unsup
    ‘matrix’: interpret the k parameter as a k by M (or length k array for one-dimensional data) array of initial c

Returns:

centroid : ndarray
    A k by N array of centroids found at the last iteration of k-means.

label : ndarray
    label[i] is the code or index of the centroid the i’th observation is closest to.