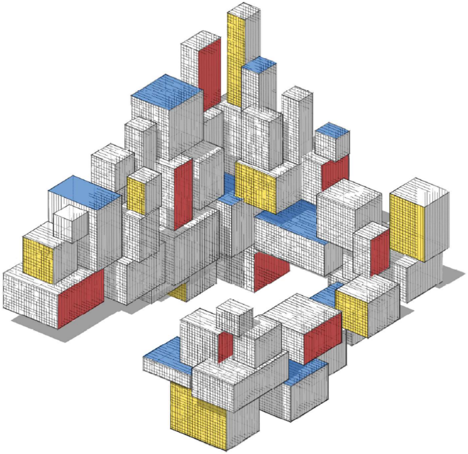
Clustering



Purpose

In this lecture we discuss:

- Mixture modeling
- Gaussian mixture
- The EM algorithm for mixture modeling

Clustering via Mixture Models

We wish to group feature vectors into clusters, such that samples within a cluster are similar to each other.

Usually, it is assumed that the number of clusters is known in advance.

Applications in data compression and storage, database searching, pattern matching, and object recognition.

A common approach is to assume that the data comes from a mixture of (usually Gaussian) distributions.

The objective is to estimate the parameters of the mixture model by maximizing the likelihood function for the data.

This is where the EM algorithm comes to the fore.

Mixture Models

Let $\mathcal{T} := \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$ be iid random vectors with values in some set $\mathcal{X} \subseteq \mathbb{R}^d$, each $\boldsymbol{X}_i$ being distributed according to the mixture density

$$g(\boldsymbol{x} \mid \boldsymbol{\theta}) = w_1 \phi_1(\boldsymbol{x}) + \cdots + w_K \phi_K(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{X}, \tag{1}$$

where $\phi_1, \ldots, \phi_K$ are pdfs on $\mathcal{X}$, and the positive weights $w_1, \ldots, w_K$ sum up to 1.

Let Z take values $1, 2, \ldots, K$ with probabilities $w_1, \ldots, w_K$, and let $\boldsymbol{X}$ be a random vector whose conditional pdf, given $Z = z$, is ϕ_z.

The joint pdf of Z and $\boldsymbol{X}$ is given by

$$\phi_{Z,\boldsymbol{X}}(z, \boldsymbol{x}) = \phi_Z(z)\, \phi_{\boldsymbol{X} \mid Z}(\boldsymbol{x} \mid z) = w_z\, \phi_z(\boldsymbol{x})$$

and the marginal pdf of $\boldsymbol{X}$ is (1).

We can interpret the latent variable Z_i as the hidden label of the cluster to which $\boldsymbol{X}_i$ belongs.

Gaussian Mixture

In a Gaussian mixture, each density ϕ_k is Gaussian with some unknown expectation vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$.

We gather all unknown parameters into a parameter vector $\boldsymbol{\theta}$.
As the components of $\mathcal{T} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$ are iid, their (joint) pdf is given by

$$g(\tau \,|\, \boldsymbol{\theta}) := \prod_{i=1}^{n} g(\boldsymbol{x}_i \,|\, \boldsymbol{\theta}) = \prod_{i=1}^{n} \sum_{k=1}^{K} w_k \, \phi_k(\boldsymbol{x}_i \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{2}$$

We can estimate $\boldsymbol{\theta}$ from an outcome τ by maximizing the log-likelihood function

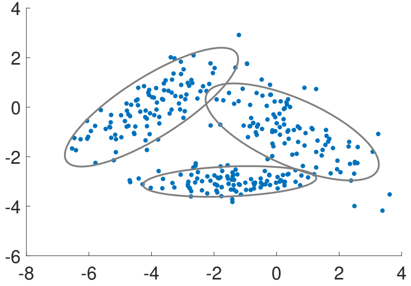
$$l(\boldsymbol{\theta} \,|\, \tau) := \sum_{i=1}^{n} \ln g(\boldsymbol{x}_i \,|\, \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \left(\sum_{k=1}^{K} w_k \, \phi_k(\boldsymbol{x}_i \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right). \tag{3}$$

However, finding the maximizer of $l(\boldsymbol{\theta} \,|\, \tau)$ is not easy in general, since the function is typically multiextremal.

Example

The data depicted below consists of 300 data points independently generated from three bivariate normal distributions

Ideally, we would like to cluster the data into three clusters.



cluster	mean vector	covariance matrix
1	$\begin{bmatrix} -4 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 2 & 1.4 \\ 1.4 & 1.5 \end{bmatrix}$
2	$\begin{bmatrix} 0.5 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 2 & -0.95 \\ -0.95 & 1 \end{bmatrix}$
3	$\begin{bmatrix} -1.5 \\ -3 \end{bmatrix}$	$\begin{bmatrix} 2 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}$

Example

A possible model for the data is to assume that the points are iid draws from an (unknown) mixture of three 2-dimensional Gaussian distributions.

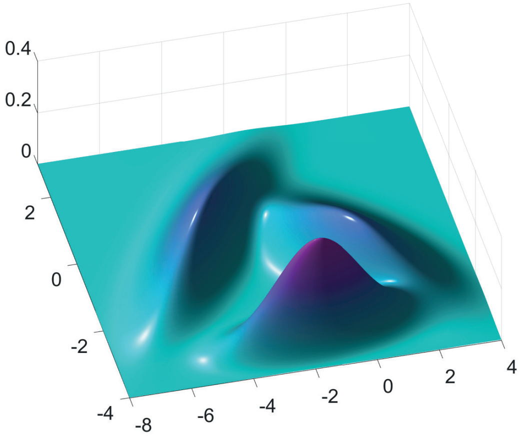


Figure: The target mixture density

EM Algorithm for Mixture Models

The joint pdf of $\mathcal{T}$ and $\boldsymbol{Z}$ is

$$g(\tau, z \,|\, \boldsymbol{\theta}) = \prod_{i=1}^{n} w_{z_i}\, \phi_{z_i}(\boldsymbol{x}_i),$$

It follows that the complete-data log-likelihood function

$$\widetilde{l}(\boldsymbol{\theta} \,|\, \tau, z) = \sum_{i=1}^{n} \ln[w_{z_i}\, \phi_{z_i}(\boldsymbol{x}_i)] \tag{4}$$

is often easier to maximize than the original log-likelihood (3).

In the E-step of the EM algorithm, the complete-data log-likelihood is replaced with $\mathbb{E}_p\, \widetilde{l}(\boldsymbol{\theta} \,|\, \tau, \boldsymbol{Z})$, where the subscript p indicates that $\boldsymbol{Z}$ is distributed according to the conditional pdf of $\boldsymbol{Z}$ given $\mathcal{T} = \tau$; that is,

$$p(z) = g(z \,|\, \tau, \boldsymbol{\theta}) \propto g(\tau, z \,|\, \boldsymbol{\theta}). \tag{5}$$

Note that $p(\boldsymbol{z})$ is of the form $p_1(z_1)\cdots p_n(z_n)$ so that, given $\mathcal{T} = \tau$, the components of $\boldsymbol{Z}$ are independent of each other.

Algorithm 1: EM Algorithm for Mixture Models

```
input: Data $\tau$, initial guess $\boldsymbol{\theta}^{(0)}$.
output: Approximation of the maximum likelihood estimate.
1 $t \leftarrow 1$
2 while a stopping criterion is not met do
3     Expectation Step: Find $p^{(t)}(\boldsymbol{z}) := g(\boldsymbol{z} \,|\, \tau, \boldsymbol{\theta}^{(t-1)})$ and
      $Q^{(t)}(\boldsymbol{\theta}) := \mathbb{E}_{p^{(t)}} \widetilde{l}(\boldsymbol{\theta} \,|\, \tau, \boldsymbol{Z})$.
4     Maximization Step: Let $\boldsymbol{\theta}^{(t)} \leftarrow \operatorname{argmax}_{\boldsymbol{\theta}} Q^{(t)}(\boldsymbol{\theta})$.
5     $t \leftarrow t + 1$
6 return $\boldsymbol{\theta}^{(t)}$
```

EM for Gaussian Mixtures

For the case of Gaussian mixtures, each $\phi_k = \phi(\cdot \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the density of a d-dimensional Gaussian distribution.

Let $\boldsymbol{\theta}^{(t-1)} = \{\{w_k^{(t-1)}\}, \{\boldsymbol{\mu}_k^{(t-1)}\}, \{\boldsymbol{\Sigma}_k^{(t-1)}\}\}$.

We first determine $p^{(t)}$ — the pdf of $\boldsymbol{Z}$ conditional on $\mathcal{T} = \tau$ — for the given guess $\boldsymbol{\theta}^{(t-1)}$.

As the components of $\boldsymbol{Z}$ given $\mathcal{T} = \tau$ are independent, it suffices to specify the discrete pdf, $p_i^{(t)}$ say, of each Z_i given the observed point $\boldsymbol{X}_i = \boldsymbol{x}_i$.

The latter can be found from Bayes' formula:

$$p_i^{(t)}(k) \propto w_k^{(t-1)} \, \phi_k(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k^{(t-1)}, \boldsymbol{\Sigma}_k^{(t-1)}), \quad k = 1, \ldots, K. \tag{6}$$

EM for Gaussian Mixtures

Next, in view of (4), the function $Q^{(t)}(\boldsymbol{\theta})$ can be written as

$$Q^{(t)}(\boldsymbol{\theta}) = \mathbb{E}_{p^{(t)}} \sum_{i=1}^{n} \Big(\ln w_{Z_i} + \ln\ \phi_{Z_i}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{Z_i}, \boldsymbol{\Sigma}_{Z_i})\Big) = \sum_{i=1}^{n} \mathbb{E}_{p_i^{(t)}} \left[\ln w_{Z_i} + \ln\ \phi_{Z_i}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_{Z_i}, \boldsymbol{\Sigma}_{Z_i})\right],$$

where the $\{Z_i\}$ are independent and Z_i is distributed according to $p_i^{(t)}$ in (6). This completes the E-step.

In the M-step we maximize $Q^{(t)}$ with respect to the parameter $\boldsymbol{\theta}$. In particular, we maximize

$$\sum_{i=1}^{n} \sum_{k=1}^{K} p_i^{(t)}(k) \left[\ln w_k + \ln \phi_k(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right],$$

under the condition $\sum_k w_k = 1$. Using Lagrange multipliers and the fact that $\sum_{k=1}^{K} p_i^{(t)}(k) = 1$ gives the solution for the $\{w_k\}$:

$$w_k = \frac{1}{n} \sum_{i=1}^{n} p_i^{(t)}(k), \quad k = 1, \ldots, K. \tag{7}$$

EM for Gaussian Mixtures

The solutions for $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ now follow from maximizing $\sum_{i=1}^n p_i^{(t)}(k) \, \ln \phi_k(\boldsymbol{x}_i \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, leading to

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^n p_i^{(t)}(k)\, \boldsymbol{x}_i}{\sum_{i=1}^n p_i^{(t)}(k)}, \quad k = 1, \ldots, K \tag{8}$$

and

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^n p_i^{(t)}(k)\, (\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^\top}{\sum_{i=1}^n p_i^{(t)}(k)}, \quad k = 1, \ldots, K. \tag{9}$$

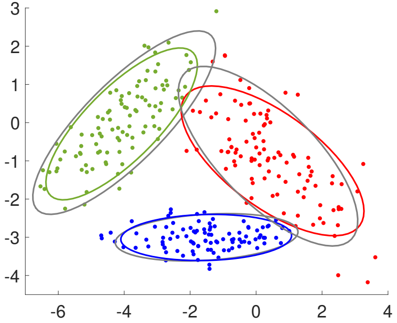
After assigning the solution parameters to $\boldsymbol{\theta}^{(t)}$ and increasing the iteration counter t by 1, the steps (6), (7), (8), and (9) are repeated until convergence is reached.

As convergence of the EM algorithm is sensitive to the initial parameters, try various different starting conditions.

Example (cont.)

The ellipses on the left-hand side of the figure show a close match between the 95% probability ellipses.

A natural way to cluster each point $\boldsymbol{x}_i$ is to assign it to the cluster k for which the conditional probability $p_i(k)$ is maximal.



weight	mean vector	covariance matrix
0.33	$\begin{bmatrix}-1.51\\-3.01\end{bmatrix}$	$\begin{bmatrix}1.75 & 0.03\\0.03 & 0.095\end{bmatrix}$
0.32	$\begin{bmatrix}-4.08\\-0.033\end{bmatrix}$	$\begin{bmatrix}1.37 & 0.92\\0.92 & 1.03\end{bmatrix}$
0.35	$\begin{bmatrix}0.36\\-0.88\end{bmatrix}$	$\begin{bmatrix}1.93 & -1.20\\-1.20 & 1.44\end{bmatrix}$

Figure: Results of the EM clustering algorithm applied to the example data.

EMclust.py

```python
import numpy as np
from scipy.stats import multivariate_normal

Xmat = np.genfromtxt('clusterdata.csv', delimiter=',')
K = 3
n, D = Xmat.shape

W = np.array([[1/3,1/3,1/3]])
M  = np.array([[-2.0,-4,0],[-3,1,-1]], dtype=np.float32)
# Note that if above *all* entries were written as integers, M would
# be defined to be of integer type, which will give the wrong answer

C = np.zeros((3,2,2))

C[:,0,0] = 1
C[:,1,1] = 1

p = np.zeros((3,300))
```

```
for i in range(0,100):

#E-step
    for k in range(0,K):
        mvn = multivariate_normal( M[:,k].T, C[k,:,:] )
        p[k,:] = W[0,k]*mvn.pdf(Xmat)

# M-Step
    p = (p/sum(p,0))   #normalize
    W = np.mean(p,1).reshape(1,3)

    for k in range(0,K):
        M[:,k] = (Xmat.T @ p[k,:].T)/sum(p[k,:])
        xm = Xmat.T - M[:,k].reshape(2,1)
        C[k,:,:] = xm @ (xm*p[k,:]).T/sum(p[k,:])
```

Clustering via Vector Quantization

Given a collection $\tau = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ of data points in some d-dimensional space $\mathcal{X}$, divide this data set into K clusters (groups) such that some loss function is minimized.

A convenient way to determine these clusters is to first divide up the entire space $\mathcal{X}$, using some distance function $\mathrm{dist}(\cdot, \cdot)$ on this space. A standard choice is the Euclidean (or L_2) distance:

$$\mathrm{dist}(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\| = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2}.$$

Distances

Other commonly used distance measures on $\mathbb{R}^d$ include the Manhattan distance:

$$\sum_{i=1}^{d} |x_i - x_i'|$$

and the maximum distance:

$$\max_{i=1,\dots,d} |x_i - x_i'|.$$

On the set of strings of length d, an often-used distance measure is the Hamming distance:

$$\sum_{i=1}^{d} \mathbb{I}\{x_i \neq x_i'\},$$

that is, the number of mismatched characters. For example, the Hamming distance between 010101 and 011010 is 4.

Voronoi Tessellation

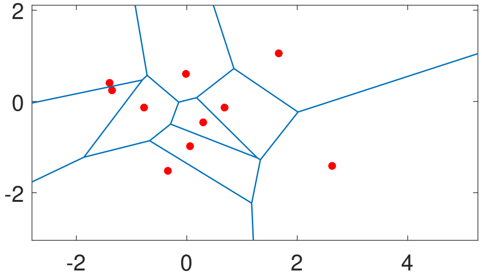
We can partition the space $\mathcal{X}$ into regions as follows: First, we choose K points $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$ called cluster centers or source vectors.

For each $k = 1, \ldots, K$, let

$$\mathcal{R}_k = \{\boldsymbol{x} \in \mathcal{X} : \mathrm{dist}(\boldsymbol{x}, \boldsymbol{c}_k) \leqslant \mathrm{dist}(\boldsymbol{x}, \boldsymbol{c}_i) \text{ for all } i \neq k\}$$

be the set of points in $\mathcal{X}$ that lie closer to $\boldsymbol{c}_k$ than any other center.

The regions or cells $\{\mathcal{R}_k\}$ divide the space $\mathcal{X}$ into a Voronoi tessellation. We used here the Euclidean distance.



Clustering According to the Closest Center

Note that here the boundaries between the Voronoi cells are straight line segments.

Once the centers (and thus the cells $\{\mathcal{R}_k\}$) are chosen, the points in τ can be clustered according to their nearest center.

The main remaining issue is how to choose the centers so as to cluster the data in some optimal way.

In terms of our learning framework, we wish to approximate a vector $\boldsymbol{x}$ via one of $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$, the vector-valued function

$$\boldsymbol{g}(\boldsymbol{x} \mid \mathbf{C}) := \sum_{k=1}^{K} \boldsymbol{c}_k \, \mathbb{I}\{\boldsymbol{x} \in \mathcal{R}_k\},$$

where $\mathbf{C}$ is the $d \times K$ matrix $[\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K]$. Thus, $\boldsymbol{g}(\boldsymbol{x} \mid \mathbf{C}) = \boldsymbol{c}_k$ when $\boldsymbol{x}$ falls in region $\mathcal{R}_k$. Within this class $\mathcal{G}$ of functions we aim to minimize the training loss.

Squared-Error Loss

For the squared-error loss, $\text{Loss}(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|^2$, the training loss is

$$\ell_{\tau_n}(\boldsymbol{g}(\cdot \,|\, \mathbf{C})) = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{g}(\boldsymbol{x}_i \,|\, \mathbf{C})\|^2 = \frac{1}{n} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{R}_k \cap \tau_n} \|\boldsymbol{x} - \boldsymbol{c}_k\|^2.$$

Thus, the training loss minimizes the average squared distance between the centers.

Most well-known clustering and vector quantization methods update the vector of centers, starting from some initial choice and using iterative (gradient-based) procedures.

This type of problem — optimization with respect to the centers — is highly multiextremal and, depending on the initial clusters, gradient-based procedures tend to converge to a *local minimum*.

K-Means

The K-means method is a simple iterative method for clustering:

- Start from an initial guess for the centers.
- Form new centers by taking sample means of the current points in each cluster (i.e., centroids).
- Repeat until convergence

At each iteration, for a given choice of centers, each point in τ is assigned to its nearest center.

A typical stopping criterion is to stop when the centers no longer change very much.

The algorithm is quite sensitive to the choice of the initial centers, so try multiple starting values.

Algorithm 2: K-Means

input: Collection of points $\tau = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, number of clusters K, initial centers $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K$.

output: Cluster centers and cells (regions).

```
1 while a stopping criterion is not met do
2     $\mathcal{R}_1, \ldots, \mathcal{R}_K \leftarrow \emptyset$ (empty sets).
3     for $i = 1$ to $n$ do
4         $\boldsymbol{d} \leftarrow [\mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{c}_1), \ldots, \mathrm{dist}(\boldsymbol{x}_i, \boldsymbol{c}_K)]$    // distances to centers
5         $k \leftarrow \mathrm{argmin}_j \, d_j$
6         $\mathcal{R}_k \leftarrow \mathcal{R}_k \cup \{\boldsymbol{x}_i\}$    // assign $\boldsymbol{x}_i$ to cluster $k$
7     for $k = 1$ to $K$ do
8         $\boldsymbol{c}_k \leftarrow \frac{\sum_{\boldsymbol{x} \in \mathcal{R}_k} \boldsymbol{x}}{|\mathcal{R}_k|}$    // compute the new center
9 return $\{\boldsymbol{c}_k\}$, $\{\mathcal{R}_k\}$
```

K-Means and the EM Algorithm

Suppose in the EM algorithm we have Gaussian mixtures with a fixed covariance matrix $\mathbf{\Sigma}_k = \sigma^2 \mathbf{I}_d$, $k = 1, \ldots, K$, where $\sigma^2 \approx 0$.

In iteration t of the EM algorithm, having obtained the expectation vectors $\boldsymbol{\mu}_k^{(t-1)}$ and weights $w_k^{(t-1)}$, $k = 1, \ldots, K$, each point $\boldsymbol{x}_i$ is assigned a cluster label Z_i according to the probabilities $p_i^{(t)}(k)$, $k = 1, \ldots, K$ given in (6).

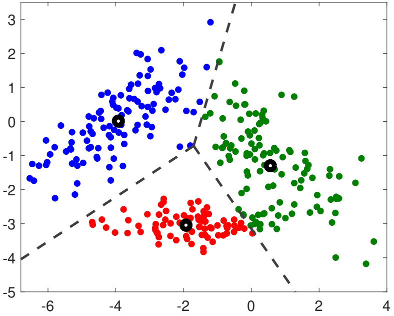
But for $\sigma^2 \to 0$ the probability distribution $\{p_i^{(t)}(k)\}$ becomes degenerate, putting all its probability mass on $\operatorname{argmin}_k \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|^2$.

This corresponds to the K-means rule of assigning $\boldsymbol{x}_i$ to its nearest cluster center.

In the M-step (8) each cluster center $\boldsymbol{\mu}_k^{(t)}$ is now updated according to the average of the $\{\boldsymbol{x}_i\}$ that have been assigned to cluster k. We thus obtain the same deterministic updating rule as in K-means.

Example: K-means Clustering

We cluster the previous example data with K-means, using the same starting centers as in the EM example.



We find the cluster centers $\boldsymbol{c}_1 = [-1.9286, -3.0416]^\top$, $\boldsymbol{c}_2 = [-3.9237, 0.0131]^\top$, and $\boldsymbol{c}_3 = [0.5611, -1.2980]^\top$.

Kmeans.py

```python
import numpy as np
Xmat = np.genfromtxt('clusterdata.csv', delimiter=',')
K = 3
n, D = Xmat.shape
c  = np.array([[-2.0,-4,0],[-3,1,-1]])  #initialize centers
cold = np.zeros(c.shape)
dist2 = np.zeros((K,n))
while np.abs(c - cold).sum() > 0.001:
   cold = c.copy()
   for i in range(0,K): #compute the squared distances
        dist2[i,:] = np.sum((Xmat - c[:,i].T)**2, 1)

   label = np.argmin(dist2,0) #assign the points to nearest centroid
   minvals = np.amin(dist2,0)
   for i in range(0,K): # recompute the centroids
      c[:,i] = np.mean(Xmat[np.where(label == i),:],1).reshape(1,2)

print('Loss = {:3.3f}'.format(minvals.mean()))
```

```
Loss = 2.288
```