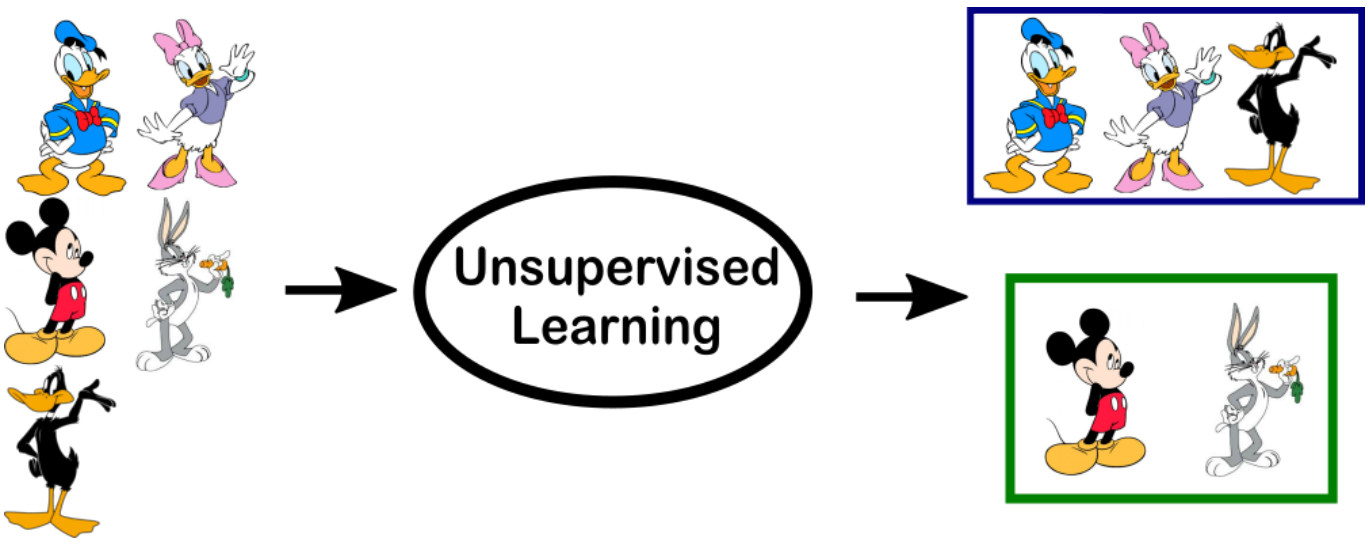
Clustering Based Unsupervised Learning

Syed Sadat Nazrul Follow
Apr 3, 2018 · 6 min read



Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from “unlabeled” data (a classification or categorization is not included in the observations). Common scenarios for using unsupervised learning algorithms include:
- Data Exploration
- Outlier Detection
- Pattern Recognition

While there is an exhaustive list of clustering algorithms available (whether you use R or Python’s Scikit-Learn), I will attempt to cover the basic concepts.

K-Means

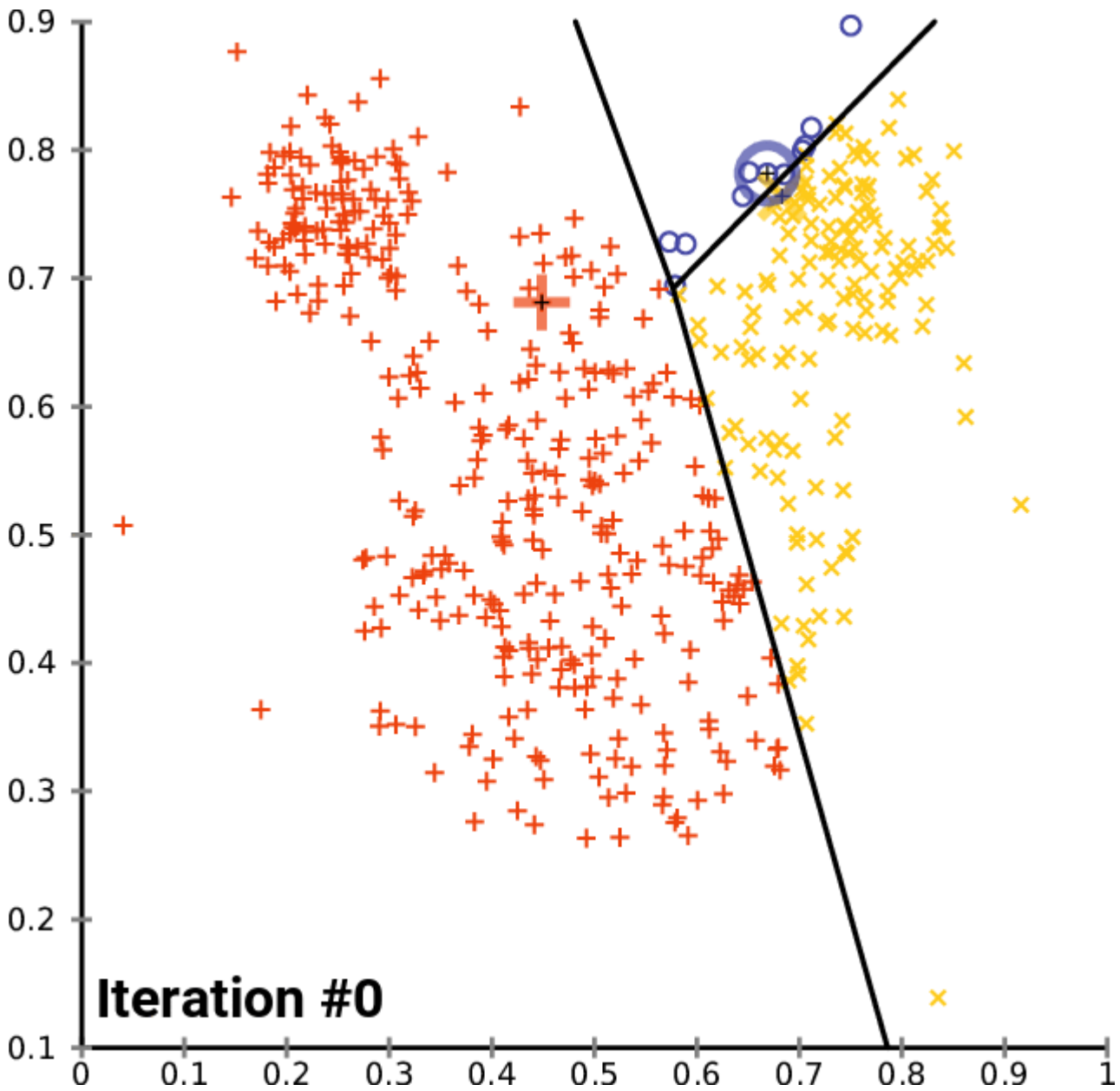
The most common and simplest clustering algorithm out there is the K-Means clustering. This algorithms involve you telling the algorithms how many possible cluster (or K) there are in the dataset. The algorithm then iteratively moves the k-centers and selects the datapoints that are closest to that centroid in the cluster.

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

   For every i, set
   $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

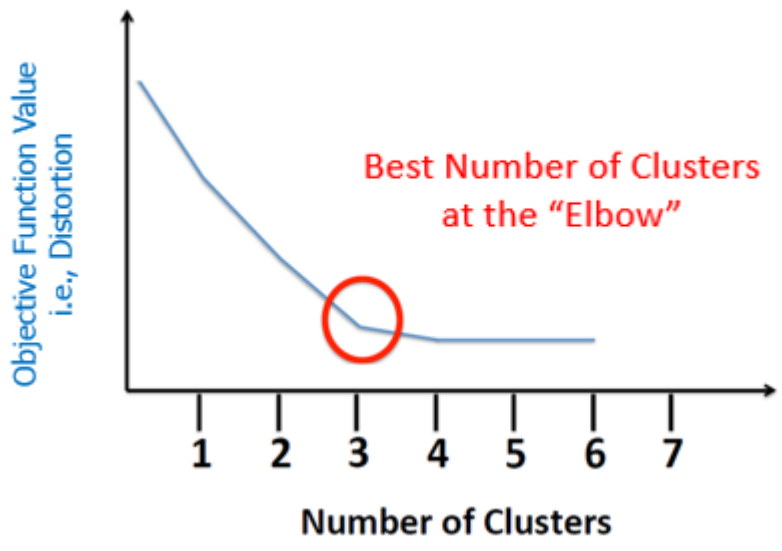
   For each j, set
   $$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

Taking K=3 as an example, the iterative process is given below:



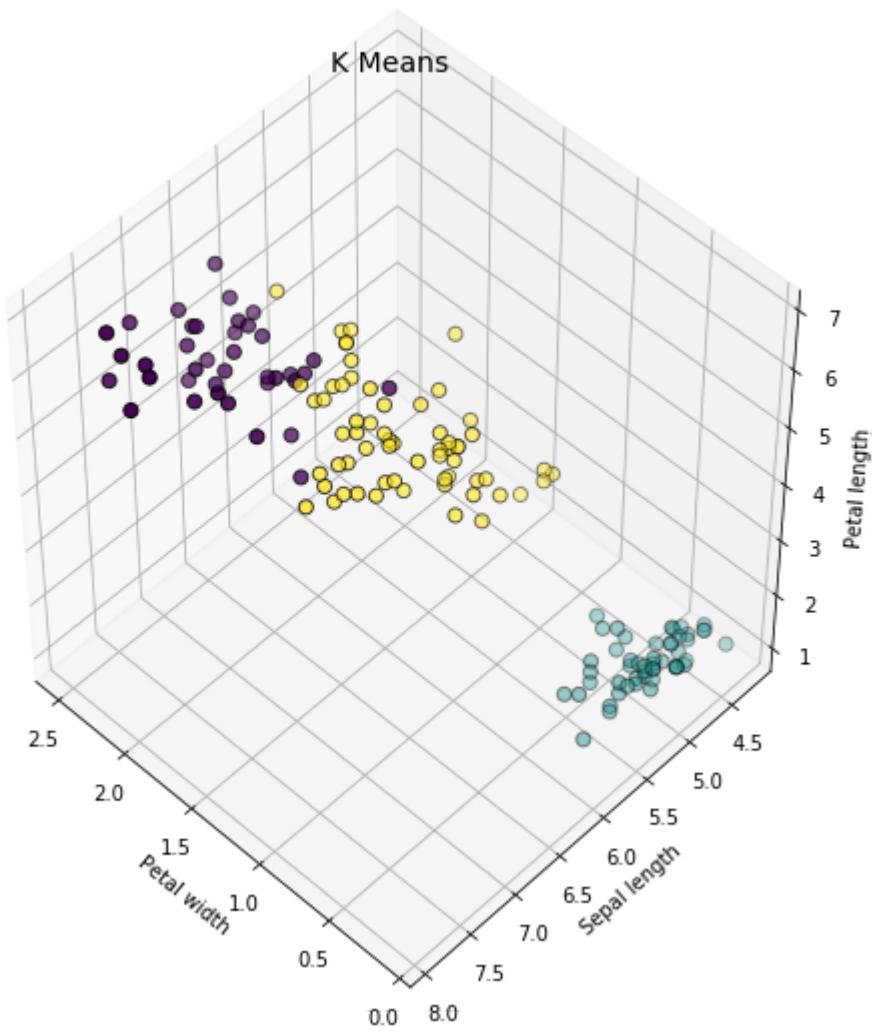
One obvious question that may come to mind is the methodology for picking the K value. This is done using an elbow curve, where the x-axis is the K-value and the y axis is some objective function. A common objective function is the average distance between the datapoints and the nearest centroid.

Elbow method



The best number for K is the "elbow" or kinked region. After this point, it is generally established that adding more clusters will not add significant value to your analysis. Below is an example script for K-Means using Scikit-Learn on the iris dataset:

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
%matplotlib inline
from sklearn import datasets


#Iris Dataset
iris = datasets.load_iris()
X = iris.data


#KMeans
km = KMeans(n_clusters=3)
km.fit(X)
km.predict(X)
labels = km.labels_


#Plotting
fig = plt.figure(1, figsize=(7,7))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
ax.scatter(X[:, 3], X[:, 0], X[:, 2],
           c=labels.astype(np.float), edgecolor="k", s=50)
ax.set_xlabel("Petal width")
ax.set_ylabel("Sepal length")
ax.set_zlabel("Petal length")
plt.title("K Means", fontsize=14)
```



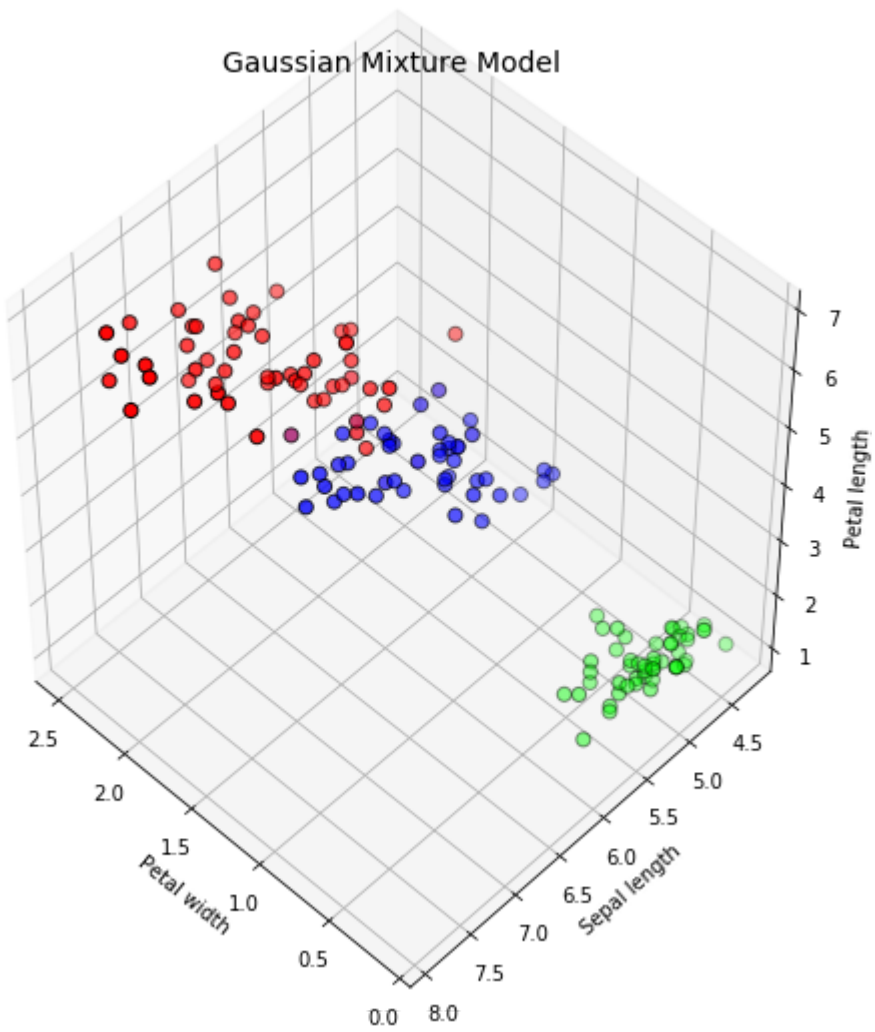
One issue with K-means, as see in the 3D diagram above, is that it does hard labels. However, you can see that datapoints at the boundary of the purple and yellow clusters can be either one. For such circumstances, a different approach may be necessary.

Mixture Models

In K-Means, we do what is called “hard labeling”, where we simply add the label of the maximum probability. However, certain data points that exist at the boundary of clusters may simply have similar probabilities of being on

eit
ins

```
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
%matplotlib inline
from sklearn import datasets


#Iris Dataset
iris = datasets.load_iris()
X = iris.data


#Gaussian Mixture Model
gmm = GaussianMixture(n_components=3)
gmm.fit(X)
proba_lists = gmm.predict_proba(X)


#Plotting
colored_arrays = np.matrix(proba_lists)
colored_tuples = [tuple(i.tolist()[0]) for i in colored_arrays]
fig = plt.figure(1, figsize=(7,7))
ax = Axes3D(fig, rect=[0, 0, 0.95, 1], elev=48, azim=134)
ax.scatter(X[:, 3], X[:, 0], X[:, 2],
           c=colored_tuples, edgecolor="k", s=50)
ax.set_xlabel("Petal width")
ax.set_ylabel("Sepal length")
ax.set_zlabel("Petal length")
plt.title("Gaussian Mixture Model", fontsize=14)
```



For the above Gaussian Mixure Model, the colors of the datapoints are based on the Gaussian probability of being near the cluster. The RGB values are based on the nearness to each of the red, blue and green clusters. If you look at the datapoints near the boundary of the blue and red cluster, you shall see purple, indicating the datapoints are close to either clusters.

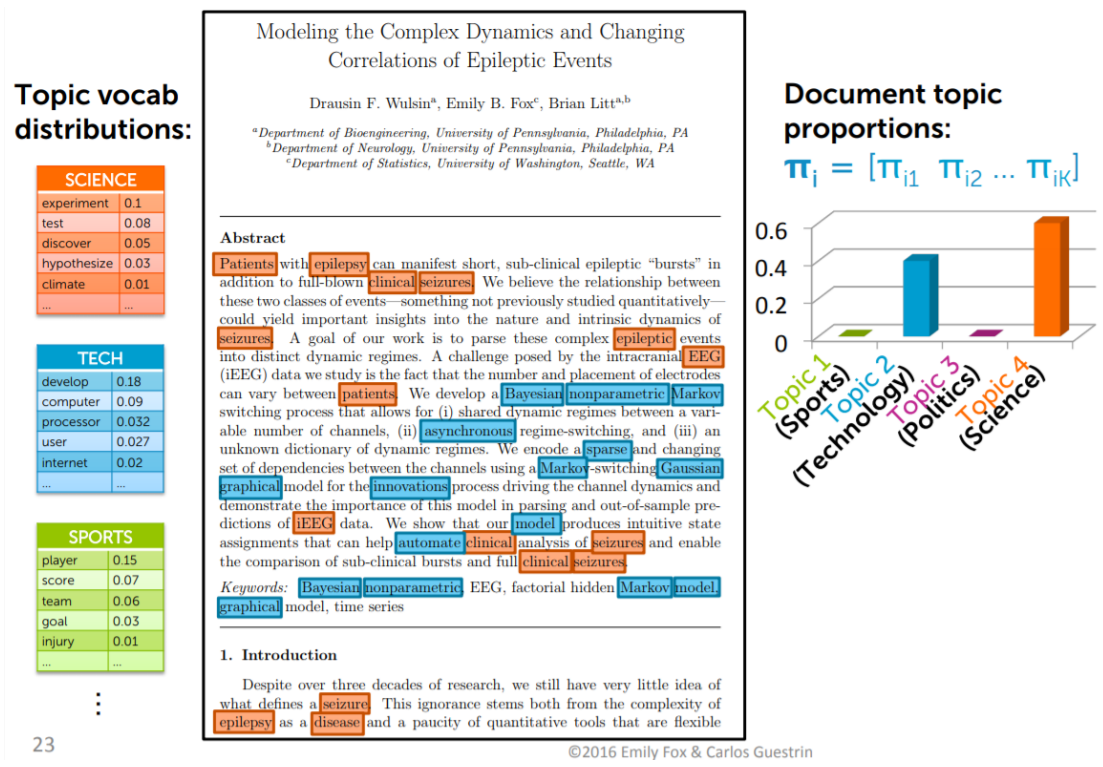
Topic Modelling

Si
ca
approach for such problems is topic modelling, where documents or words in a document are categorized into topics. The simplest of these is the TF-IDF model. The TF-IDF model classifies words based on their importance. This is determined by how frequent are they in specific documents (e.g. specific science topics in scientific journals) and words that are common among all documents (e.g. stop words).

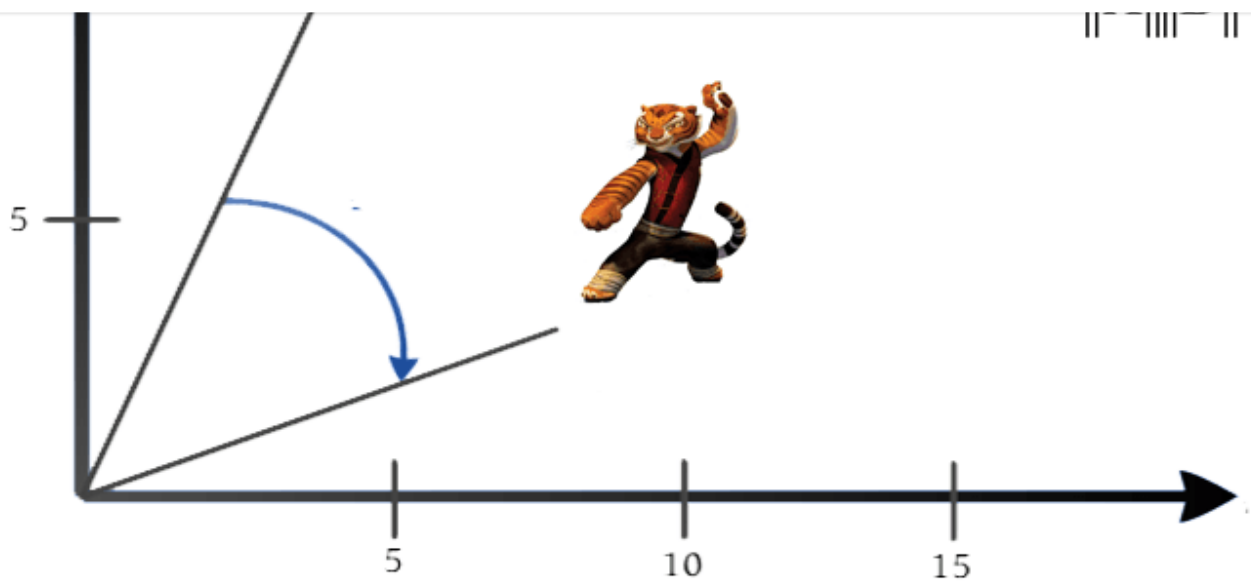
$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

One of my favorite algorithms is the Latent Dirichlet Allocation or LDA model. In this model, each word in the document is given a topic based on the entire document corpus. Below, I have attached a slide from the University of Washington's Machine Learning specialization course:

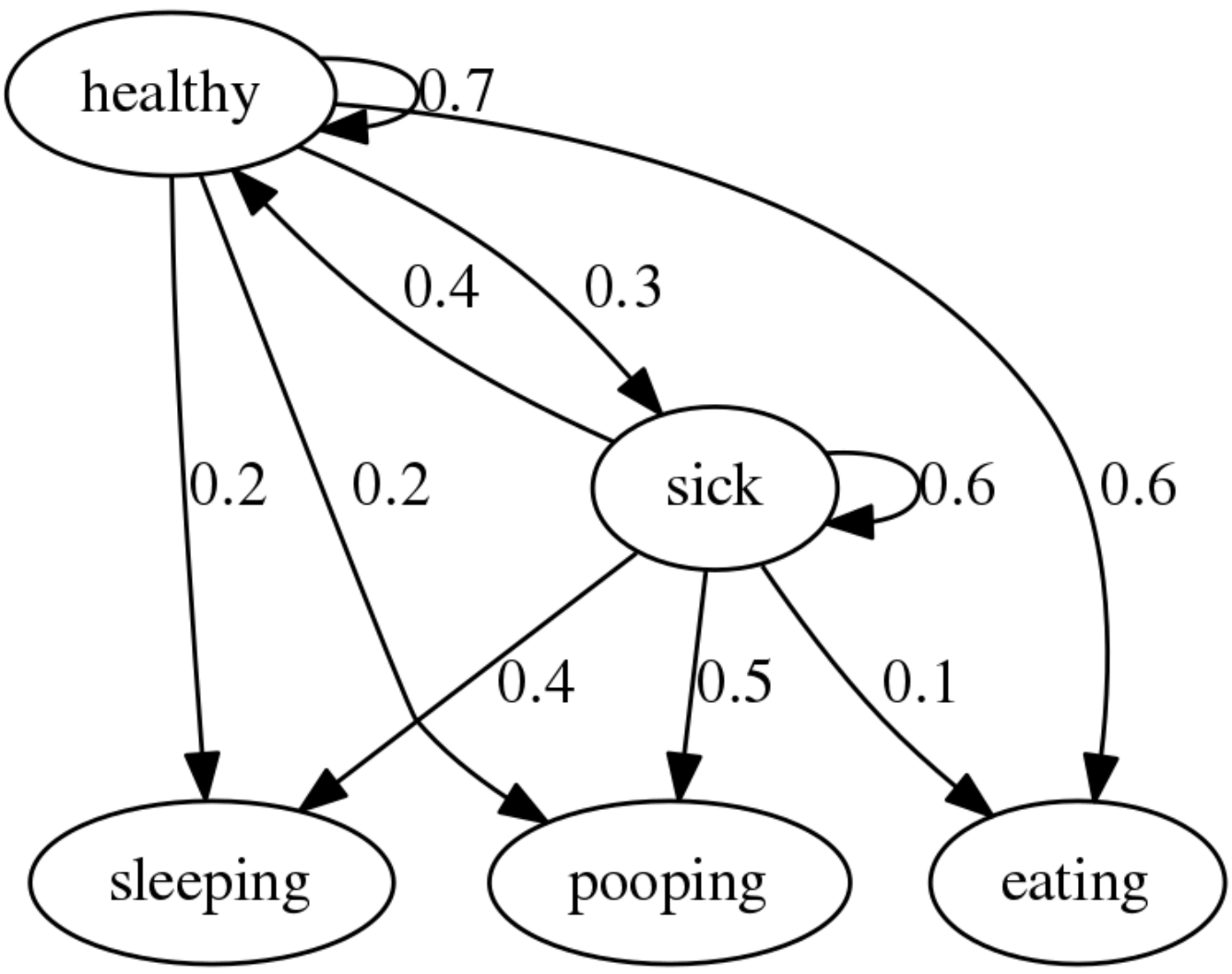


The mechanics behind the LDA model itself is hard to explain in this blog. However, a common question people have is deciding on the number of topics. While there is no established answer for this, personally I prefer to implement a elbow curve of K-Means of the word vector of each document. The closeness of each word vector can be determined by the cosine distance.



Hidden Markov Model

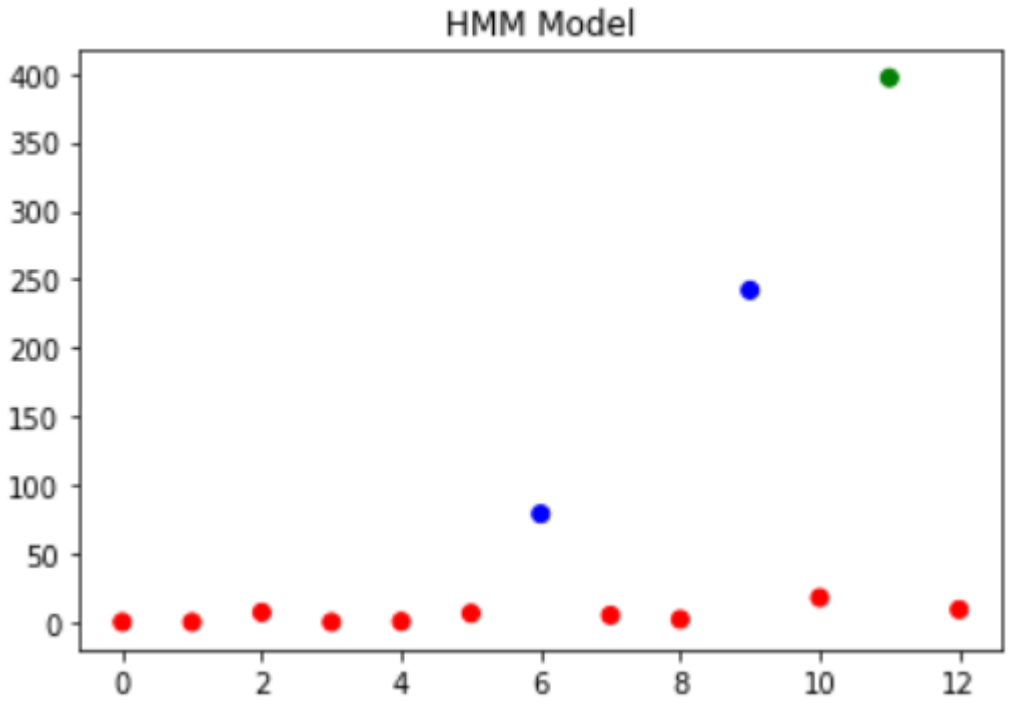
Finally, let's cover some timeseries analysis. For clustering, my favourite is using Hidden Markov Models or HMM. In a Markov Model, we look for states and the probability of the next state given the current state. An example below is of a dog's life in Markov Model.



Let's assume the dog is sick. Given the current state, there is a 0.6 chance it will continue being sick the next hour, 0.4 that it is sleeping, 05 pooping, 0.1 eating and 0.4 that it will be healthy again. In an HMM, you provide how many states there may be inside the timeseries data for the model to compute. An example of the Boston house prices dataset is given below with 3 states.

```
from hmmlearn import hmm
import numpy as np
```

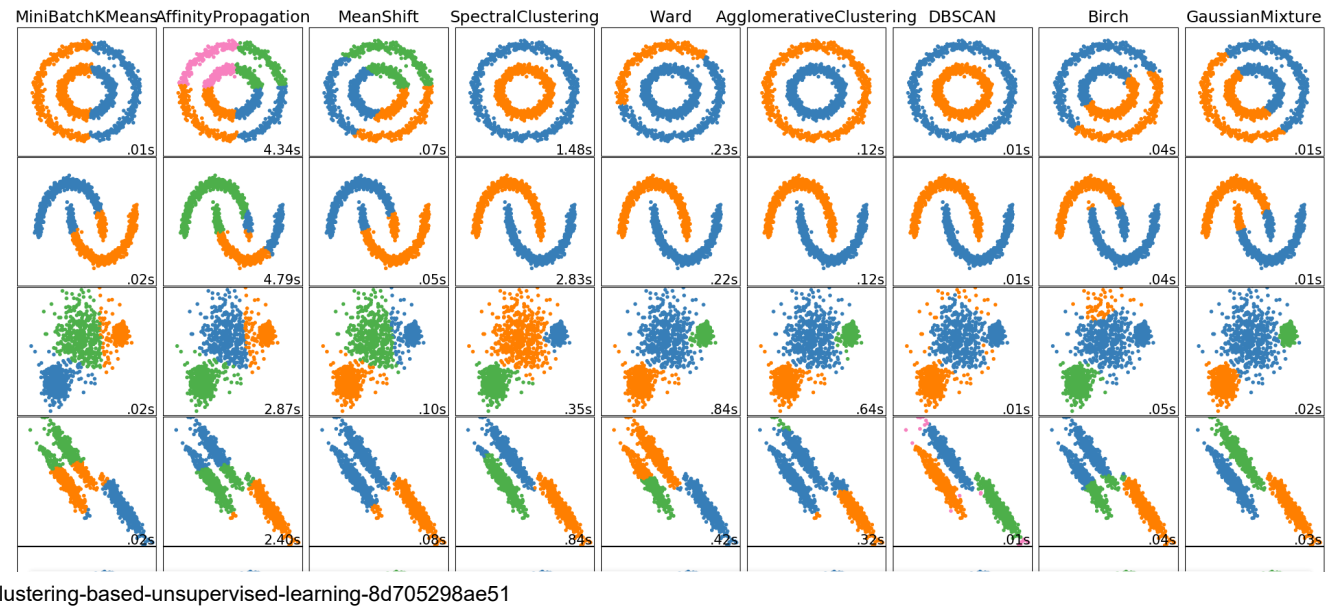
```
#Data
boston = datasets.load_boston()
ts_data = boston.data[1,:]


#HMM Model
gm = hmm.GaussianHMM(n_components=3)
gm.fit(ts_data.reshape(-1, 1))
states = gm.predict(ts_data.reshape(-1, 1))


#Plot
color_dict = {0:"r",1:"g",2:"b"}
color_array = [color_dict[i] for i in states]
plt.scatter(range(len(ts_data)), ts_data, c=color_array)
plt.title("HMM Model")
```

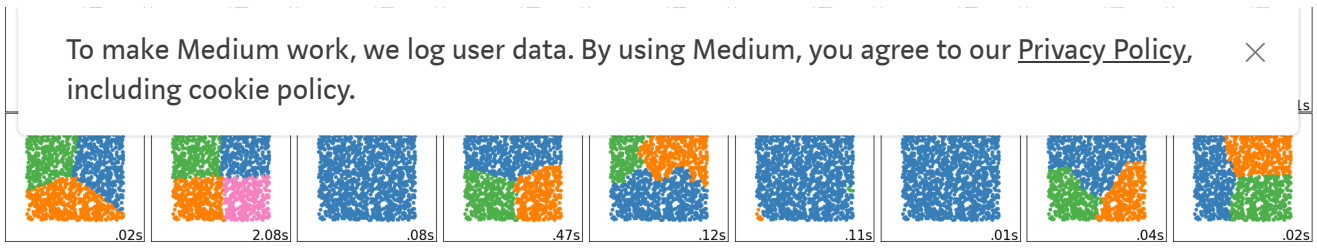


As with every clustering problem, deciding the number of states is also a common issue. This may either be domain based. e.g. in voice recognition, it is common practice to use 3 states. Another possibility is using an elbow curve.

Final Thoughts

As I have mentioned at the beginning of this blog, it is not possible for me to cover every single unsupervised models out there. At the same time, based on your use case, you may need a combination of algorithms to get a different perspective of the same data. With that I would like to leave you off with Scikit-Learn's famous clustering demonstrations on the toy dataset:





Unsupervised Learning Clustering K Means Lda Topic Modeling