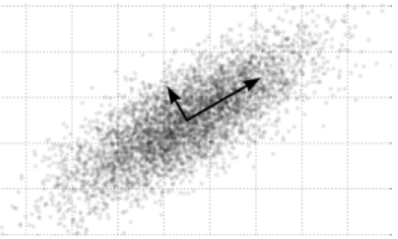
Clustering

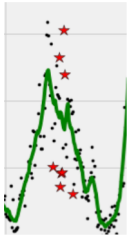
Unsupervised learning

Terence Parr
MSDS program
University of San Francisco

Unsupervised learning overview

- In a nutshell, we have just X not $X \rightarrow y$ and would like to know about X, such as density or interesting subregions/vectors of X (n x p matrix)
- Clustering
  - k-means / k-medioid / mean-shift
  - hierarchical clustering
  - spectral clustering
- Recommendation engines
  - collaborative filtering ("other people like you bought X")
  - market basket analysis / association rules ("what do people buy together?); see *a priori algorithm*
- Principle components analysis (orthogonal vectors of most variation)
- Page rank for ranking most important articles/nodes
- Word embeddings like glove (matrix factorization)
- Anomaly detection (fraud or network attack detection)





See Chapter 14 in ESLII

The problem is…

- All of those unsupervised methods can be useful
- But a big problem is that there no clear measure of success, such as the metrics used by supervised learning
- E.g., you have bank transactions and no idea which are fradulent; design algorithm to identify fraud; now, how do you know if your algorithm works?

"Almost all of AI's recent progress is through one type, in which some input data (A) is used to quickly generate some simple response (B)."
Andrew Ng in *What Artificial Intelligence Can and Can't Do Right Now*
Harvard Business Review November 9, 2016

Clustering overview

- Sounds awesome but useful only in limited circumstances, such as vector quantization / compression and usually only when p is small
- Is it really what we want anyway?
  - Imagine clustering a customer db into 4 clusters; now what?
  - You have 4 groups of, say, 4 million records each; what do you do with it?
  - Let's say you can identify spending groups like "technerd", "shoeshopper", et… What do you do with that info?
  - Old joke: You know you're wasting half of your marketing money, but you don't know which half!
  - Can try to market to those groups but don't we really want to know what kind of ad people click on? Run an ad campaign and track customer->clicks; now you have a supervised problem

Clustering preliminaries

- Each $x^{(i)}$ in X is a point in p space with p coordinates
- Space can be Euclidean but categorical vars present challenges
- All such spaces must have distance($x^{(i)}$, $x^{(j)}$) measure
- L_1 and L_2 are common distances for Euclidean space
- L_∞ also useful: max abs difference in any dimension
- For large p and/or binary values, better to use cosine distance (angle between 2 vectors); cos(θ)=$\frac{v \cdot w}{\|v\|\|w\|}$ so acos(θ) is distance
- Two flavors: point-assignment and agglomerative

Distance measure requirements

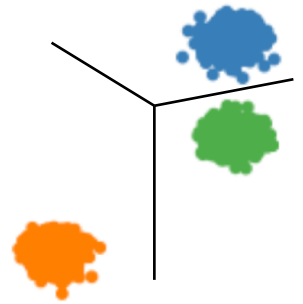
1. Always nonnegative; only distance(v,v) is 0
2. Symmetry; distance(v,w) = distance(w,v)
3. Triangle inequality; distance(v,w)+distance(w,z)≥distance(v,z)

From http://www.mmds.org/; see it for more on distance metrics

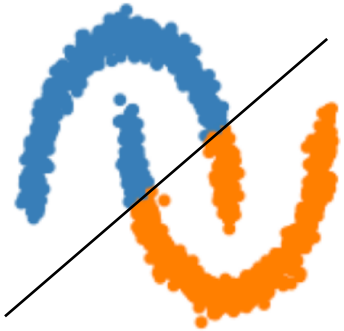
k-means clustering

- Assumes Euclidean space
- Clusters separated by straight lines only
- User provides X and number of clusters to find, k
- Idea is to pick k centroids in p space and assign points to cluster with closest centroid then recompute centroids
- Repeat until the cluster assignments stop changing
- Can select k points as initial centroids:
  - At random (seems to do a crappy job for large p)
  - By picking k distant points (*k-means*++ is a variation for initial selection)
- Algorithm converges using Euclidean distance
- Not guaranteed to find optimal clusters

k-means works

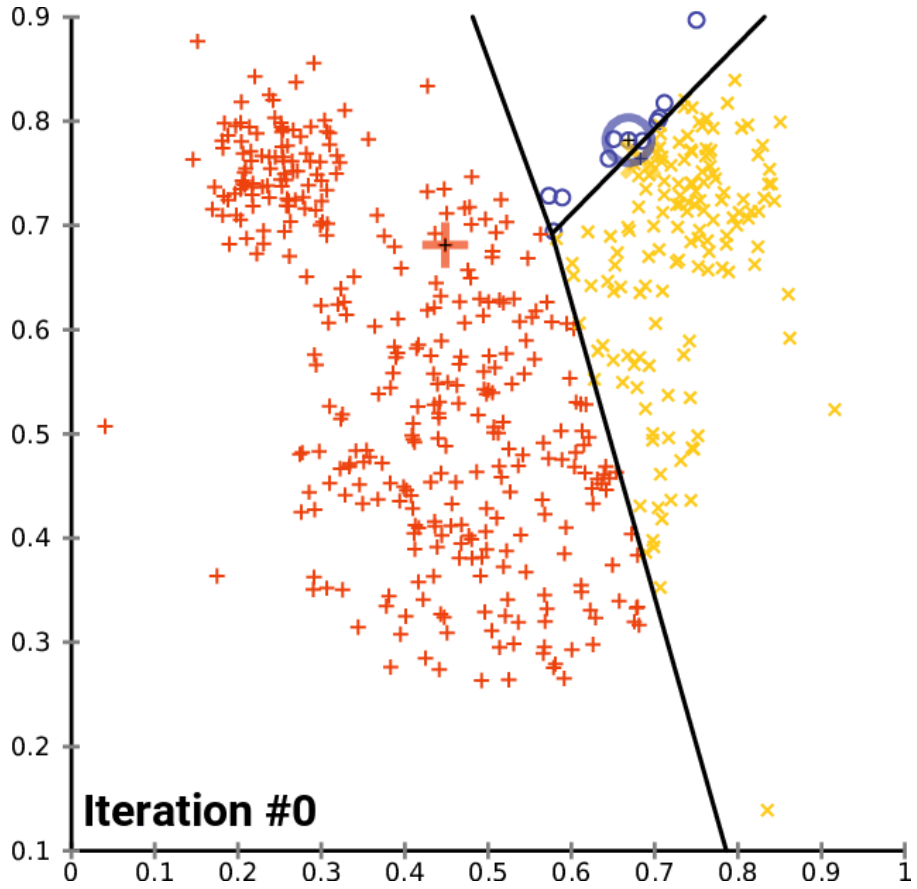


k-means fails



k-mean clustering animation

- k-means gives Voronoi tessellation
- It assumes that all points in the cluster are contiguous; sounds obvious, but for most real problems this assumption doesn't hold
- If true clusters are noncontiguous, k-means will give poor results



Animation from https://en.wikipedia.org/wiki/K-means_clustering

k-means algorithm

1 **Algorithm:** *kmeans(X,k)*

2 Select k points from X as initial centroids $m_{1..k}^{(t=0)}$ for clusters $C_{1..k}^{(t=0)}$

3 **repeat**

4 **foreach** $x \in X$ **do**

5 $i = \arg\min_j distance(x, m_j^{(t)})$ *(find closest centroid to x)*

6 Add x to cluster $C_i^{(t+1)}$ *(assign x to cluster)*

7 **for** $i = 1..k$ **do**

8 $m_i^{(t+1)} = \frac{1}{|C_i^{(t+1)}|} \sum_{x \in C_i^{(t+1)}} x$ *(recompute centroids)*

9 **end**

10 **end**

11 **until** $C_{1..k}^{(t+1)} = C_{1..k}^{(t)}$ *(until clusters don't change)*

k-means application: MNIST

(Known digits so we can measure error)

- Goal: cluster MNIST digit greyscale images

```
df_digits = pd.read_csv("data/mnist-10k-sample.csv")
images = df_digits.drop('digit', axis=1)
kmeans = KMeans(n_clusters=10) # k=10 digits
kmeans.fit(df_subset.values)
y_pred = kmeans.predict(df_subset)
correct = np.sum(df_subset['digit']!=y_pred)
```

- Results vary a lot but close to perfect score depending on initial cluster centroid selection; p=28x28=784 pixels/image
- Somehow clusters are pretty contiguous, so k-means works ok

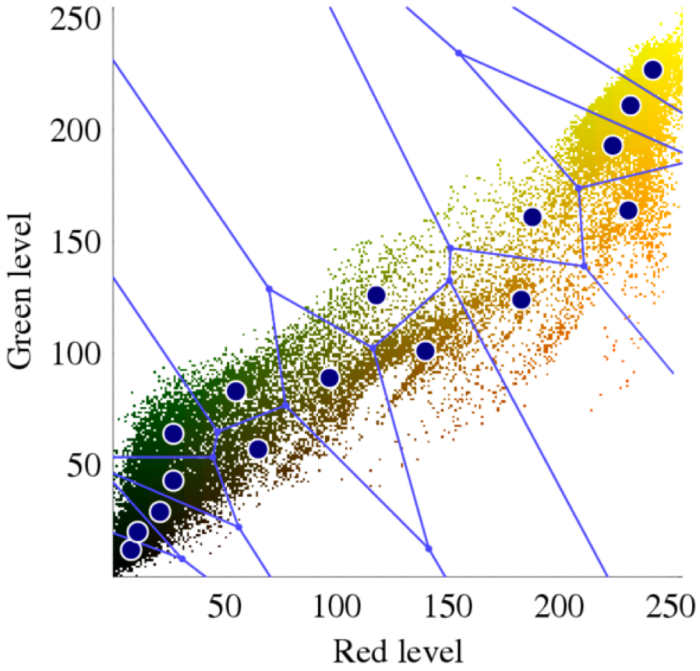
k-means application: Breast cancer

(Known cancer/benign target so we can measure error)

- k-means (with sklearn using k-means++ centroid selector) is bad
- Does about 50% accuracy with huge stddev; i.e., doesn't work
- Dimensions are p=30, which could be part of the issue but most likely true clusters have noncontiguous regions; k-means will fail in this situation

k-means application: color quantization

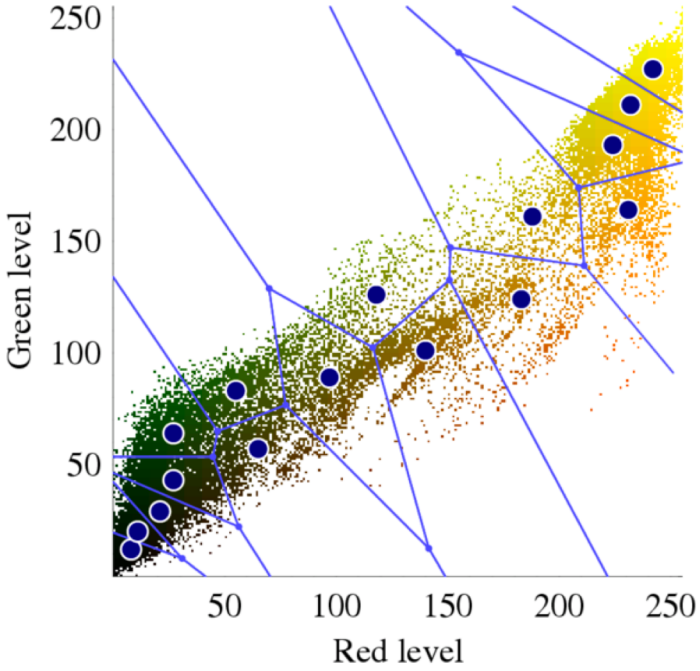
- Color pictures typically use lots of unique colors, possibly 10s of thousands
- Each pixel in the image has Red/Green/Blue colors, 1 byte per RGB = 3 bytes (24 bits)
- Each RGB is a 3D coordinate of color: (R, G, B), with possibly tens of thousands of unique combinations
- Example with just red/green (omit blue):



See https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
Image from https://en.wikipedia.org/wiki/Color_quantization

Color quantization cont'd

- (R,G,B) takes 3 bytes per pixel which makes images really big
- If a picture only has 256 unique colors we can map all (R,G,B) vectors to a single byte; the color "index" 0..255 points into a color palette with the full (R,G,B) vectors; 3x compression for each pixel so a massive compression
- If picture has more than 256, we can cluster in RGB space with k=256 and it will group similar colors together; then we pick the centroid as the colors in the palette



Color quantization example

Original image
(96,615 colors)

Quantized image
(10 colors, k-Means)

Quantized image
(10 colors, at random)

See https://github.com/parrt/msds621/blob/master/notebooks/clustering/kmeans.ipynb

Confusion point

- k-means' centroids don't have to be points in X, usually isn't
- *k-medians* uses median not mean for centroids and, thus, minimizes w.r.t. L1 not L2 distance; median even for single dimension doesn't have to be point in $x^{(i)}$ space
- *k-medoids* (not spelled *k-medioids*) requires medoids to be points in X; works with any distance measure; sounds like k-means but algorithm pretty different; gotta pick "centrally located point"

Trouble with k-means

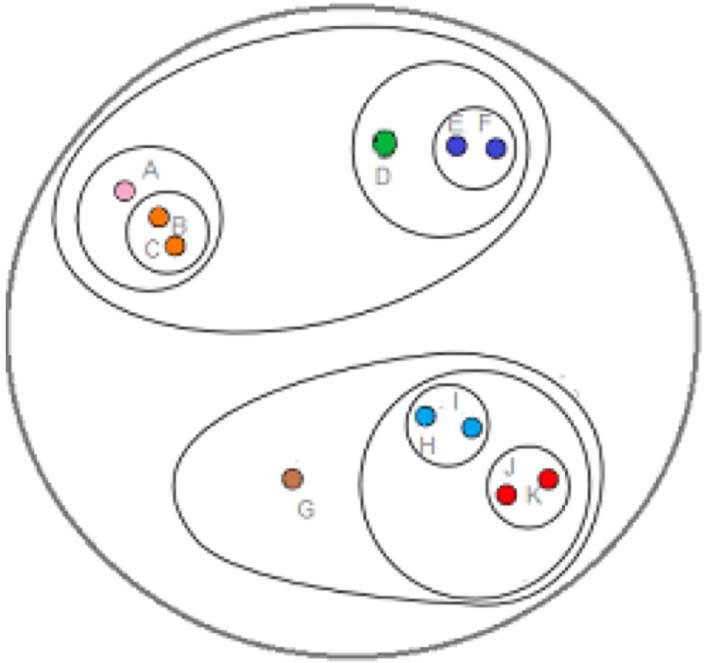
- k-means requires user give number of clusters k
- Picking k is usually a problem
- Color quantization and MNIST digits have known k, but few do

Hierarchical (agglomerative) clustering

- Idea: put every point into its own singleton cluster; repeatedly group two closest clusters into a meta-cluster until there is a single cluster left
- Can also stop when distance between clusters sufficiently large
- We need a cluster distance metric; simplest is just the distance between cluster centroids; called the *linkage criterion*
- Result is a tree of clusters, one cluster per level
- We get all possible clusters up to k=n-1 for n observations

Dendograms

- Dendogram is a tree of clusters, one cluster per level
- Distance from node to children reflects between-cluster-metric



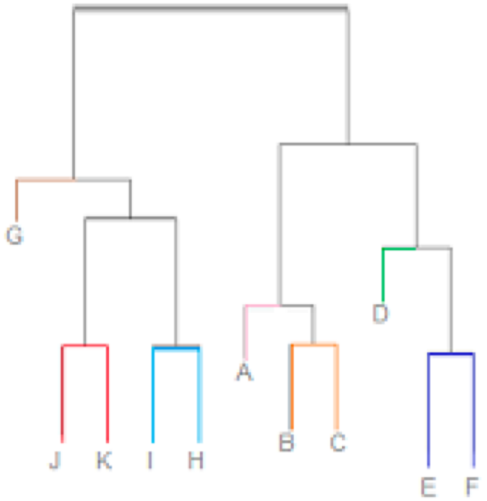
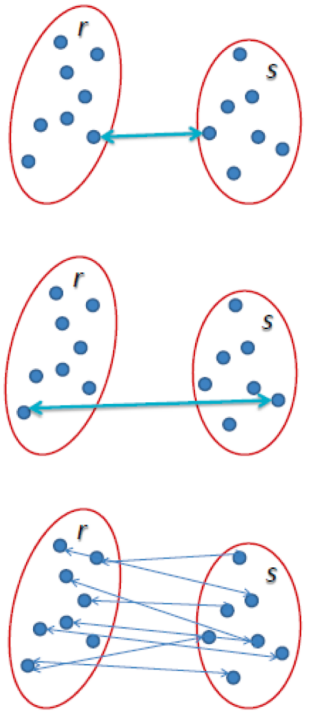


Image from https://www.statisticshowto.datasciencecentral.com/hierarchical-clustering/

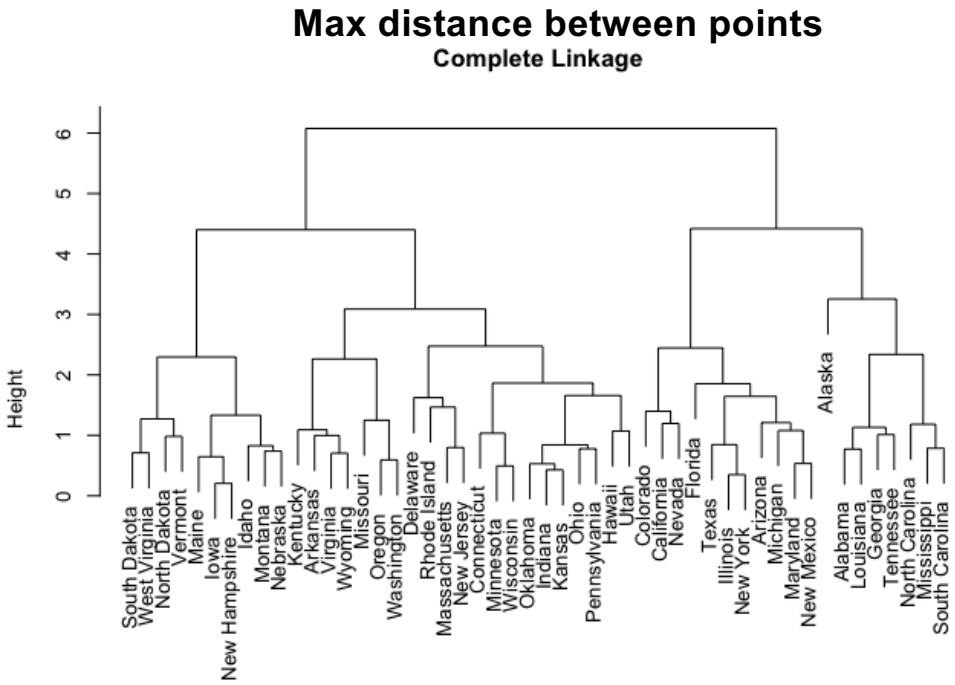
Between-cluster distance metrics (*linkage*)

1. Minimum distance between any two points in the cluster (*single linkage*); tends not to get compact clusters and can get chains of points
2. Max distance between point pairs (*complete linkage*); tends to get compact clusters but points can be closer to other clusters than those within their cluster
3. Average distance of all point pairs from two clusters (*group average linkage*); tries to get compact clusters that are far apart



Images from http://saedsayad.com/clustering_hierarchical.htm

Effect of between-cluster-metric



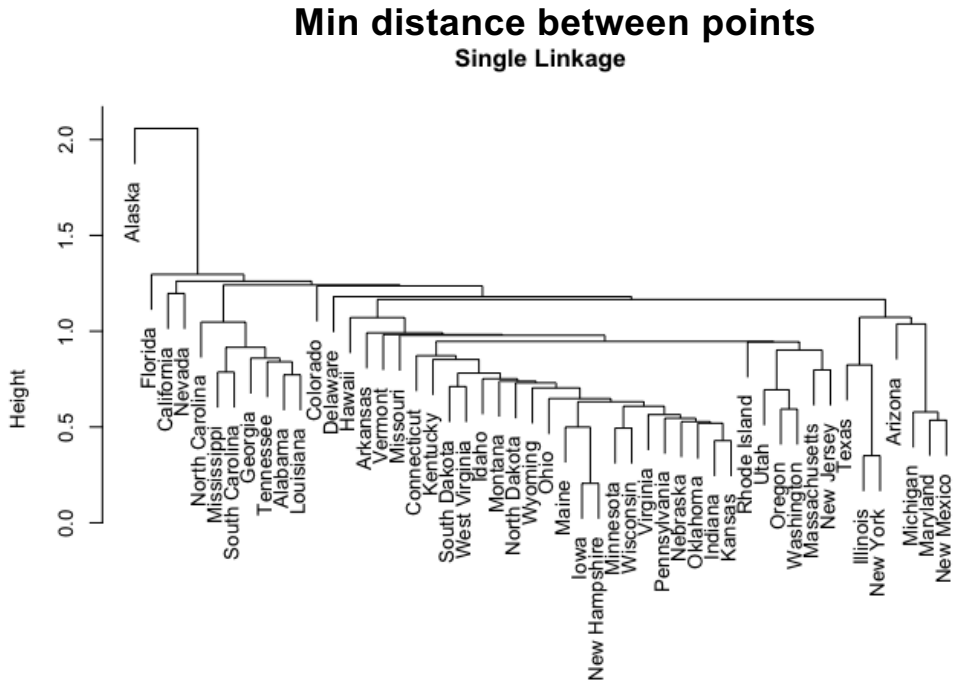


Image from https://uc-r.github.io/hc_clustering

UNIVERSITY OF SAN FRANCISCO

Non-numeric clustering

- How do you cluster documents?
- Can use edit distance or Jaccard similarity between text docs
- But what about tabular data with nominal categorical variables?
- In non-numeric space, what is a centroid vector?
- There are similarity measures for categoricals, but I'm not a big fan, particularly with mixed numeric and categorical data

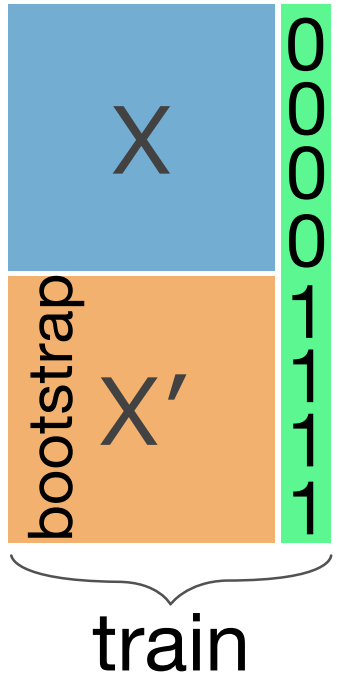
Breiman's RF distance metric

- Goal: similarity($x^{(i)}, x^{(j)}$) or distance($x^{(i)}, x^{(j)}$) for any two feature vectors in X, ($x^{(i)}, x^{(j)}$), even in the presence of mixed categorical and numeric data
- Random Forests to the rescue again with clever trick that turns unsupervised into the supervised problem then drives similarity matrix between all $x^{(i)}, x^{(j)}$ pairs
- Proximity matrix: count how often $x^{(i)}, x^{(j)}$ appear in same leaf; normalize by number of leaves
- Use 1 minus proximity to get distance, then can use any clustering algorithm we want like k-means, …

See https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Random Forest clustering mechanics

1. Consider X as class 0
2. Duplicate and bootstrap columns of X to get X': class 1
   - Breiman: X' created by "…*sampling at random from the univariate distributions…*" of X
   - X' destroys relationships between columns of X
3. Create y to label X vs X'
4. Train RF on stacked [X,X'] → y
5. Walk all leaves of all trees, bumping proximity[i,j] for all $x^{(i)}, x^{(j)}$ pairs in leaf; divide proximity by number of leaves
6. Cluster using 1-proximity for distance matrix



Breiman's RF distance X' from X

Here's how to create X' from X

```
def df_scramble(X : pd.DataFrame) -> pd.DataFrame:
    X_rand = X.copy()
    for colname in X:
        X_rand[colname] = \
            np.random.choice(X[colname].unique(),
                             len(X),
                             replace=True)
    return X_rand
```

Breiman's RF distance conjure supervised from unsupervised

```
def conjure_twoclass(X : pd.DataFrame)\
                    -> (pd.DataFrame, pd.Series):
    X_rand = df_scramble(X)
    X_synth = pd.concat([X, X_rand], axis=0)
    y_synth = np.concatenate([np.zeros(len(X)),
                              np.ones(len(X_rand))],
                             axis=0)
    return X_synth, pd.Series(y_synth)
```