Data Science

Yogesh Kulkarni

KNN

K-Nearest Neighbors (KNN)

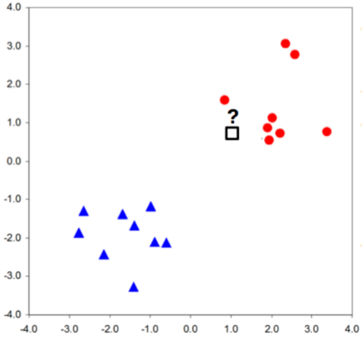
- Supervised Classification.
- Algorithm stores all available cases and classifies new cases by a majority vote of its k neighbors.

K-Nearest Neighbors (KNN)

- "A man is known by the company he keeps"
- KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

Intuition for kNN

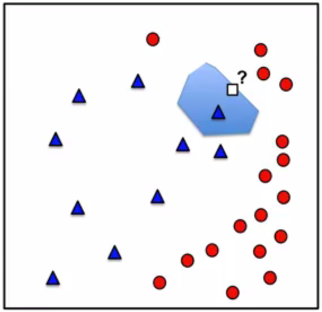
- Set of points (x,y) and two classes (red/blue)
- Is the box red or blue?
- How did you guess? By Decision Tree? or by SVM?
- You saw, nearby points are red.



Neighborhood is the basis of KNN.

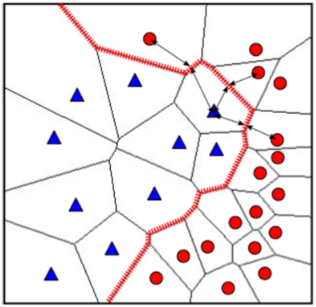
Nearest-neighbor classification

- Use the intuition to classify a new point x: find the most similar training example x'; predict its class y'
- This is true for all the points shown in blue patch/cell.
- Every training example is going to have a small cell around it.
- All cells together are called as Voronoi Diagram.
- Examples: Post office region demarcation, Utility stations, etc.



Nearest-neighbor classification

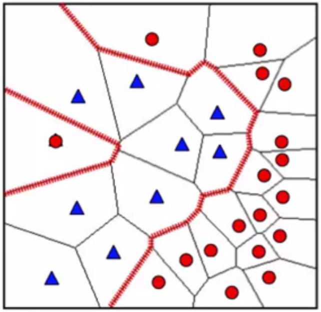
- Each cell is such that points in it are closer to its owner Centroid than any other Centroids.
- Each cell corresponds to a training example.
- Classification boundary separates the cells based on classes
- See how a Non-linear complex partitioning is possible, than just linear or margin separation



Nearest-neighbor classification

Limitations

- But, if you have a Outlier, say the red shown, then whole classification goes for a toss.
- Boundary becomes too complicated.
- So, Sensitive to Outliers.
- Remedy: to look at more than one neighbors to classify yourself.
- Majority Voting. Adds stability.



Scaling Issues

- Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
- Example, with three dimensions
  - height of a person may vary from 1.5m to 1.8m
  - weight of a person may vary from 90lb to 300lb
  - income of a person may vary from $10K to $1M
- Income will dominate if these variables aren't standardized.

Standardization

- Treat all features "equally" so that one feature doesn't dominate the others
- Common treatment to all variables
  - Standardize each variable: Mean = 0, Standard Deviation = 1
  - Normalize each variable: Max = 1, Min = 0

KNN Algorithm

Classification Algorithm

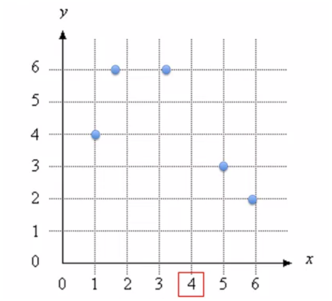
- Given: training examples (x_i, y_i) and test point x to classify.
- Here, there is no traditional training phase, where model is built.
- Compute distance $D(x, x_i)$ to every training example x_i
- Select k closest instances $x_{i1} \dots x_{ik}$ and their labels $y_{i1} \dots y_{ik}$
- Output the class $y*$ which is most frequent in $y_{i1} \dots y_{ik}$

KNN as Regression

- Given: training examples (x_i, y_i) with y_i as real valued, like profits, price, etc. and test point x to predict the target value.
- Here is no traditional training phase, where model is built.
- Compute distance $D(x, x_i)$ to every training example x_i
- Select k closest instances $x_{i1} \dots x_{ik}$ and their labels $y_{i1} \dots y_{ik}$
- Output the target $y*$ which is MEAN of $y_{i1} \dots y_{ik}$

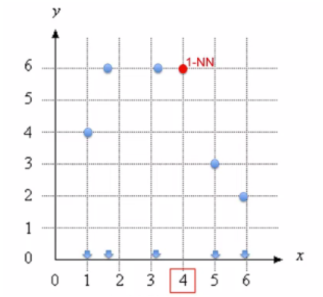
KNN Regression Example: 1d

- Given:Some set of points (x_i, y_i). The function/relation between them is unknown.
- Goal: given, unseen x, find its y value.



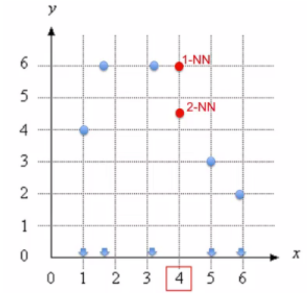
KNN Regression Example: 1d

- Who are the neighbours of $x = 4$ on x axis?
- If we want only one nearest neighbour (1-NN) then 3.2 is nearest, its y is 6, so thats the final prediction for $x = 4$.



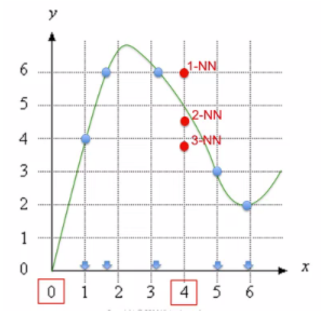
KNN Regression Example: 1d

- If we find 2-NN then 3.2 and 5 are the neighbours
- Their ys are 6 and 3.
- Avereage of theose ys is 4.5. So for $x = 4$, y is predicted as 4.5



KNN Regression Example: 1d

- Its easy to see that if you go on adding neighbours, you may not be more accurate all the time.
- If you take all the neighbours?
- As it is just avarage of those many y values aorund.
- Is that good? No. There is one sweet spot (tradeoff).

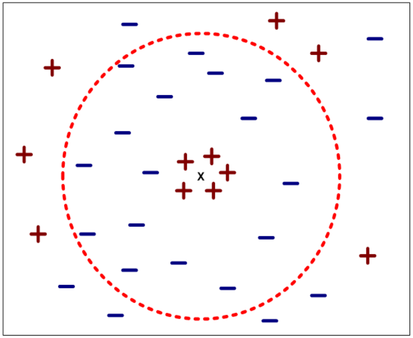


KNN Regression Example: 1d

- So, if you are in the middle, then its intrapolation, works.
- If you are at boundaries then, its hard to predict. Extrapolation
- Find for $x = 0$ with 1-NN, 2NN.

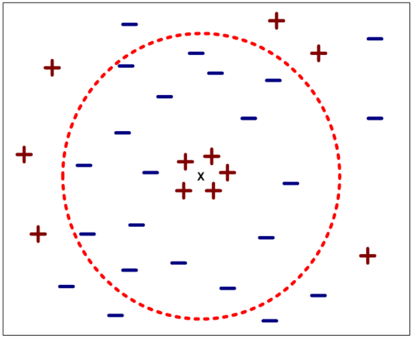
Choosing the right k

- Say, we are classifying middle point, into either +s or -s
- Depending on k (ie number of neighbours), it may get classified as + or -
- Say if k is upto 5, then it would be +, if say,20 then it would be -ve



Choosing the right k

- If k is too small, sensitive to noise points in the training data
- Susceptible to overfitting
- If k is too large, neighborhood may include points from other classes
- Susceptible to misclassification



Choosing the right k

- Do, cross validation.
- Take validation set out of training set. Make it separate, not part of Training set any more.
- Try different k's on the validation set.
- You will predict y for evey x in the test set.
- As you know the correct labels, see which k gives best result.

Similarity Criterion - Distance

Distance measures

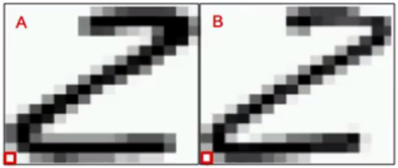
- Crucial component
- Finds similarity value between two data points.
- Types: Euclidean (for floats), Hamming (for enums), Minkowski (general form)

Distances

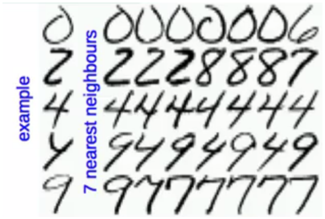
- The case being assigned to the class is most common among its K nearest neighbors measured by a distance function.
- These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance.

Example

- Handwritten digit recognition
- 16×16 bitmaps
- Euclidean Distance over pixels between testing bitmap A and training bitmap B. $D(A, B) = \sqrt{\sum_r \sum_c (A_{r,c} - B_{r,c})^2}$
- Find 7 nearest training examples, their known digit labels.
- Find most occurring among those labels



Example



- For “0”, max vote goes to “0”.
- For “2”, there is a tie, may add more neighbors to decide.
- For “4”, things may fail.
- Simple algorithm, but impressive results.
- Accuracy:
  - 7-NN: 95.2%
  - SVM: 95.8%
  - Humans: 97.5%, Humans are not perfect either!!

Distance measures (Recap)

- Euclidean is for numerical features: $\sqrt{\sum (x_d - x'_d)^2}$
- Hamming is for categorical features: $\sum 1_{x_d \neq x'_d}$ ie number of attributes, where x, x' differ
- Minkowski distance: $p\sqrt{\sum (x_d - x'_d)^p}$
  - p = 2: Euclidean
  - p = 1: Manhattan
  - p = 0: Hamming

Implementing KNN

Writing our Own KNN from Scratch

A machine learning algorithm usually consists of 2 main steps:

- A training step that takes as input the training data X and the corresponding target y and outputs a learned model h
- A predict step that takes as input new and unseen observations and uses the function h to output their corresponding responses.

Writing our Own KNN from Scratch

In the case of KNN, which as discussed earlier, is a lazy algorithm, the training block reduces to just memorizing the training data

```
def train(X_train, y_train):
  # do nothing
  return
```

Writing our Own KNN from Scratch

Now we need to write the predict method which must do the following:

- It needs to compute the euclidean distance between the “new” observation and all the data points in the training set.
- It must then select the K nearest ones and perform a majority vote.
- It then assigns the corresponding label to the observation.

Writing our Own KNN from Scratch

```
def predict(X_train, y_train, x_test, k):
  distances = []
  targets = []

  for i in range(len(X_train)):

    distance = np.sqrt(np.sum(np.square(x_test -
                                                  X_train[i, :])))
    distances.append([distance, i])

  distances = sorted(distances)

  for i in range(k):
    index = distances[i][1]
    targets.append(y_train[index])

  return Counter(targets).most_common(1)[0][0]
```

Writing our Own KNN from Scratch

Putting it all together, we can define the function KNearestNeighbor, which loops over every test example and makes a prediction.

```python
def kNearestNeighbor(X_train,y_train,X_test, predictions, k):
  train(X_train, y_train)

  for i in range(len(X_test)):
    predictions.append(predict(X_train, y_train,
                                                       X_test[i, :], k))
```

Writing our Own KNN from Scratch

Let's go ahead and run our algorithm

```
# making our predictions
predictions = []

kNearestNeighbor(X_train, y_train, X_test, predictions, 7)

# transform the list into an array
predictions = np.asarray(predictions)

# evaluating accuracy
accuracy = accuracy_score(y_test, predictions)
print('\nThe accuracy is %d%%' % accuracy*100)
```

Optimizing KNN

- Comparing a query point a in d dimensions against n training examples computes with a runtime of $O(nd)$, which can cause lag as points reach millions or billions.
- Popular choices to speed up KNN include:
  - Vernoi Diagrams: partitioning plane into regions based on distance to points in a specific subset of the plane
  - Grid Indexes: carve up space into d-dimensional boxes or grids and calculate the NN in the same cell as the point
  - Locality Sensitive Hashing (LSH): abandons the idea of finding the exact nearest neighbors. In- stead, batch up nearby points to quickly find the most appropriate bucket B for our query point.

Summary

Final Thoughts on Nearest Neighbors

- Nearest-neighbors classification is part of a more general technique called instance-based learning
- Use specific instances for prediction, rather than a model
- Nearest-neighbors is a lazy learner
- Performing the classification can be relatively computationally expensive
- No model is learned up-front

Final Thoughts on Nearest Neighbors

Pros

- The most attractive features of the K-nearest neighbor algorithm is that is simple to understand and easy to implement.
- With zero to little training time, it can be a useful tool for off-the-bat analysis of some data set you are planning to run more complex algorithms on.
- Furthermore, KNN works just as easily with multiclass data sets whereas other algorithms are hardcoded for the binary setting.
- Finally, as we mentioned earlier, the non-parametric nature of KNN gives it an edge in certain settings where the data may be highly "unusual".

Final Thoughts on Nearest Neighbors

Cons

- Computationally expensive testing phase which is impractical in industry settings.
- Can suffer from skewed class distributions. For example, if a certain class is very frequent in the training set, it will tend to dominate the majority voting of the new example (large number = more common).
- Finally, the accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.

K- Nearest Neighbors (Recap)

- k = parameter, chosen by analyst
- For a given test instance, use the k “closest” points (nearest neighbors) for performing classification
- “closest” points: defined by some proximity metric, such as Euclidean Distance
- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of k, the number of nearest neighbors to retrieve

K-Nearest Neighbors with Scikit-Learn

K-Nearest Neighbors

```
# k-Nearest Neighbor
from sklearn import datasets
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
# load iris the datasets
dataset = datasets.load_iris()
# fit a k-nearest neighbor model to the data
model = KNeighborsClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

(Ref: Machine Learning Algorithm Recipes in scikit-learn - Jason Brownlee)

Clustering

The Problem

- Imagine a storekeeper who keeps a record of all his customers' purchase histories.
- This allows him to look up the type of products an individual buyer might be interested in.
- However, doing this for each individual is grossly inefficient.
- A better solution would be to categorize his customers into groups, with each group having similar preferences.
- This would allow him to reach out to more customers with each product recommendation.

The Questions

The problem is, the storekeeper does not know:

- How his customers should be categorized?
- How many of such categories exist?

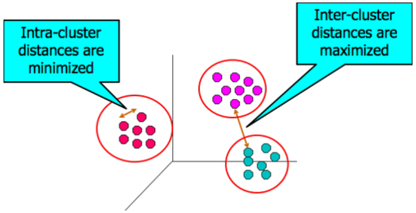
Clustering

So, how do I discover natural groupings or segments in my data?

- Often we are given a large mass of data with no training labels.
- So we have no prior idea what to look for.
- We can get some insight to get started, looking for similar items or "clusters"

Clustering

- Organizing data into classes such that there is
  - High intra-class similarity
  - Low inter-class similarity
- More informally, finding natural groupings among objects.


Intra-cluster distances are minimized
Inter-cluster distances are maximized

(Reference: Introduction to Data Mining - Tan, Steinbach, Kumar)

Types of Clustering

Partition vs. Hierarchical

- Partition Clustering: A division of data into non-overlapping clusters, such that each data object is in exactly one subset
- Hierarchical Clustering: A set of nested clusters organized as a hierarchical tree
  - Each node (cluster) is union of its children (subclusters)
  - Root of tree: cluster containing all data objects
  - Leaves of tree: singleton clusters

Complete vs. Partial

- Complete Clustering: Every object is assigned to a cluster
- Partial Clustering: Not every object needs to be assigned
  - Motivation: some objects in a dataset may not belong to well-defined groups
  - Noise, outliers, or simply "uninteresting background" data

Exclusive vs. Non-exclusive

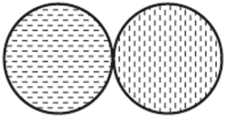
- Exclusive Clustering: Assignment is to one cluster
- Non-Exclusive Clustering: Data objects may belong to multiple clusters
  - Motivation: multi-class situations
  - Example: "student employee"

Well-Separated Clusters: any point in a cluster is closer to every other point in the cluster than to any point not in the cluster

(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.

Center-based Clusters

- an object in a cluster is closer to the center of a cluster than to the center of any other cluster
- Center of a cluster ("the most central point"):
    - Centroid: the mean of all the points in the cluster (usually for continuous attributes)
    - Medoid: the most "representative" point of a cluster (usually for categorical attributes)

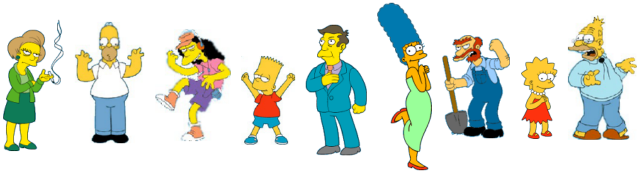


(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.

Clustering Aspects

Clustering

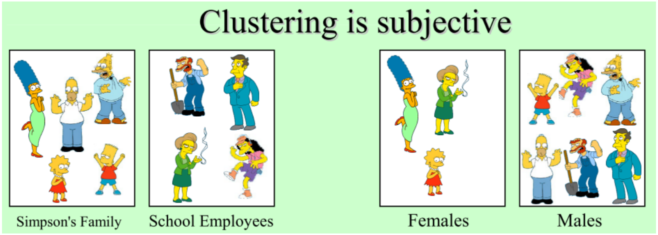
Notion of a Cluster can be Ambiguous



(Reference: K means Clustering Algorithm - Kasun Ranga Wijeweera)

Clustering

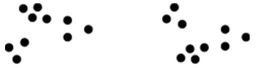
Notion of a Cluster can be Ambiguous



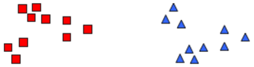
(Reference: K means Clustering Algorithm - Kasun Ranga Wijeweera)

Clustering

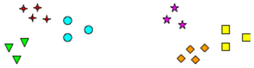
Notion of a Cluster can be Ambiguous

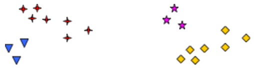

How many clusters?

By the human visual system, it looks like two clusters.


Two Clusters

But it really depends on the characteristics of the data.
These clustering may not be unreasonable:


Six Clusters


Four Clusters

K-means

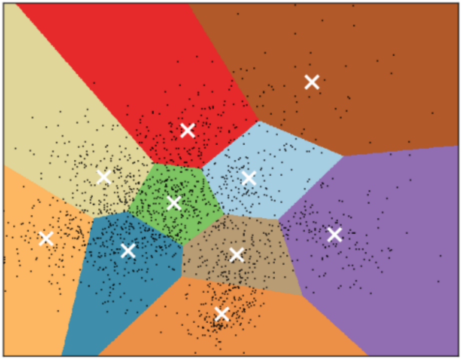
Clustering

- K-Means Clustering is one of the many techniques that constitute "unsupervised learning".
- Creating a labeling of objects with cluster (class) labels
- But, these labels are derived exclusively from the data.

K-Means Clustering

- Formally: a method of vector quantization
- Partition space into Voronoi cells
- Separate samples into n groups of equal variance
- Uses the Euclidean distance metric

K-Means Clustering



Clustering by K-Means

- It operates by computing the "mean" of some attributes.
- That "mean" then becomes the center of one cluster.
- Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).
- Data points inside a cluster are homogeneous and heterogeneous to peer groups.

How to cluster?

- Clustering is different than prediction
- Dividing data into groups (clusters) in some meaningful or useful way
- Clustering should capture “natural structure of the data”

Clustering Algorithms

- Clustering is different than prediction
- K-nearest neighbors is very different from K-means, although both are somewhat based on 'Similarity' or 'Distance'
- It is a classification (or regression) algorithm that in order to determine the classification of a point,
- Decides class of the test case based on the classes of the K nearest points.
- It is supervised because you are trying to classify a point based on the known classification of other points.)

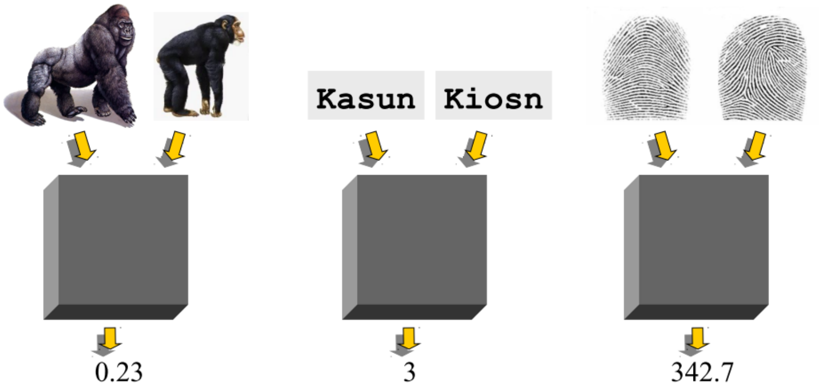
How to cluster?

- How to measure “proximity”?
- Proximity: similarity or dissimilarity between two objects
- Similarity: numerical measure of the degree to which two objects are alike

Proximity - Distance Measures

Defining Distance Measure/MetricS

Say, some black box calculations.



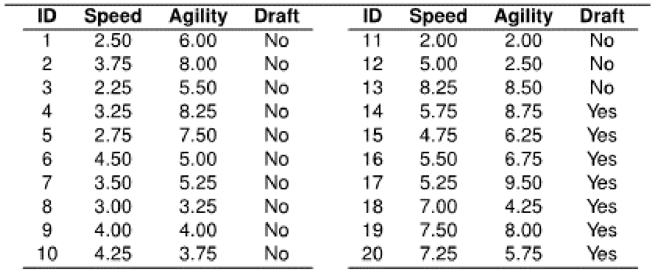
Some distance formulas. (Reference: K means Clustering Algorithm - Kasun Ranga Wijeweera)

Dissimilarities between Data Objects

- A common measure for the proximity between two objects is the Euclidean Distance
- Defined for one-dimension, two-dimensions, three-dimensions, any n-dimensional space
- Generically: Minkowski Distance Metric
  - $r = 1$: L_1 norm: Manhattan or Taxi cab distance
  - $r = 2$: L_2 norm: Euclidean distance

Example

Table: The speed and agility ratings for 20 college athletes labelled with the decisions for whether they were drafted or not.

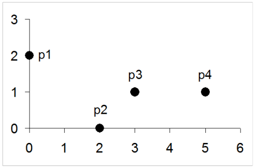


ID	Speed	Agility	Draft	ID	Speed	Agility	Draft
1	2.50	6.00	No	11	2.00	2.00	No
2	3.75	8.00	No	12	5.00	2.50	No
3	2.25	5.50	No	13	8.25	8.50	No
4	3.25	8.25	No	14	5.75	8.75	Yes
5	2.75	7.50	No	15	4.75	6.25	Yes
6	4.50	5.00	No	16	5.50	6.75	Yes
7	3.50	5.25	No	17	5.25	9.50	Yes
8	3.00	3.25	No	18	7.00	4.25	Yes
9	4.00	4.00	No	19	7.50	8.00	Yes
10	4.25	3.75	No	20	7.25	5.75	Yes

$$d(id_{12}, id_5) = \sqrt{(5.00 - 2.75)^2 + (2.50 - 7.50)^2} = \sqrt{30.0625} = 5.4829$$

Distance Matrix

- Once a distance metric is chosen, the proximity between all of the objects in the dataset can be computed
- Can be represented in a distance matrix
- Pairwise distances between points



L1 norm distance. "Manhattan" Distance.

L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

L2 norm distance. Euclidean Distance.

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

K-means Algorithm

How K-means forms cluster?

Algorithm

- Iterative refinement
- Three basic steps
- Step 1: Choose k
- Iterate over:
  - Step 2: Assignment
  - Step 3: Update centroids
- Repeats until convergence has been reached

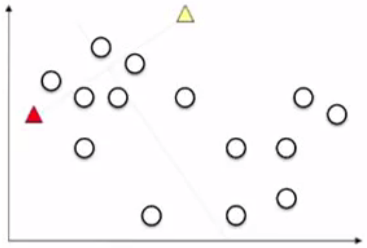
How K-means forms cluster?

Algorithm

1. K-means picks k points known as centroids.
2. Each data point finds distances with each centroid and goes to the one who is closest.
3. In the next iteration, it computes a new centroid of each cluster based on the latest cluster members.
4. As we have new centroids, repeat step 2 and 3.
5. Repeat this process until convergence occurs i.e. centroids do not change.

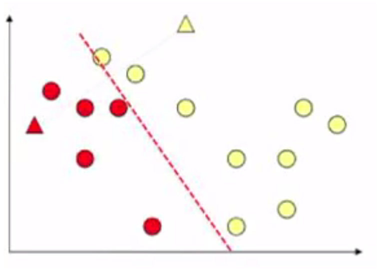
Algorithm: Steps

- Data points are K=2 given.
- So random two points are chosen as centroids



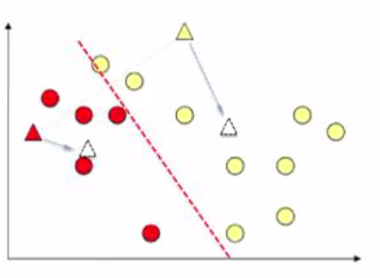
Algorithm: Steps

- Each data point is assigned to closest centroid
- Closeness is based on Distance



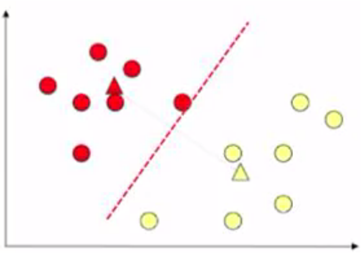
Algorithm: Steps

- Calculate new centroids of respective groups
- By averaging NUMERIC variables only.



Algorithm: Steps

- Repeat.
- Memberships may change now.
- Update till centroids don't move appreciably.
- Clustering is DONE.



K-Means Implementation

K-means Algorithm : From Scratch

Load data

```
data = loadmat('data/ex7data2.mat')
X = data['X']
```

K-means Algorithm : From Scratch

A function that finds the closest centroid for each instance in the data:

```python
def find_closest_centroids(X, centroids):
    m = X.shape[0]
    k = centroids.shape[0]
    idx = np.zeros(m)

    for i in range(m):
        min_dist = 1000000
        for j in range(k):
            dist = np.sum((X[i,:] - centroids[j,:]) ** 2)
            if dist < min_dist:
                min_dist = dist
                idx[i] = j

    return idx
```

K-means Algorithm : From Scratch

Test

```
initial_centroids = np.array([[3, 3], [6, 2], [8, 5]])

idx = find_closest_centroids(X, initial_centroids)

idx[0:3]
array([ 0.,  2.,  1.])
```

K-means Algorithm : From Scratch

A function to compute the centroid of a cluster:

```
def compute_centroids(X, idx, k):
    m, n = X.shape
    centroids = np.zeros((k, n))

    for i in range(k):
        indices = np.where(idx == i)
        centroids[i,:] = (np.sum(X[indices,:], axis=1) /
                    len(indices[0])).ravel()
    return centroids

compute_centroids(X, idx, 3)
array([[ 2.42830111,  3.15792418],
       [ 5.81350331,  2.63365645],
       [ 7.11938687,  3.6166844 ]])
```

K-means Algorithm : From Scratch

We can init centroids using some random sample of data-points as well

```
def init_centroids(X, k):
    m, n = X.shape
    centroids = np.zeros((k, n))
    idx = np.random.randint(0, m, k)
    for i in range(k):
        centroids[i,:] = X[idx[i],:]
    return centroids

init_centroids(X, 3)
array([[ 1.15354031,  4.67866717],
       [ 6.27376271,  2.24256036],
       [ 2.20960296,  4.91469264]])
```

K-means Algorithm : From Scratch

Core

```python
def run_k_means(X, initial_centroids, max_iters):
    m, n = X.shape
    k = initial_centroids.shape[0]
    idx = np.zeros(m)
    centroids = initial_centroids

    for i in range(max_iters):
        idx = find_closest_centroids(X, centroids)
        centroids = compute_centroids(X, idx, k)

    return idx, centroids
```

K-means Algorithm : From Scratch

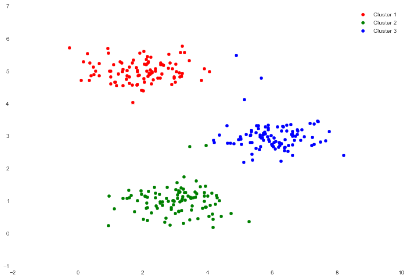
Run

```
idx, centroids = run_k_means(X, initial_centroids, 10)

cluster1 = X[np.where(idx == 0)[0],:]
cluster2 = X[np.where(idx == 1)[0],:]
cluster3 = X[np.where(idx == 2)[0],:]
```

K-means Algorithm : From Scratch

Plot:

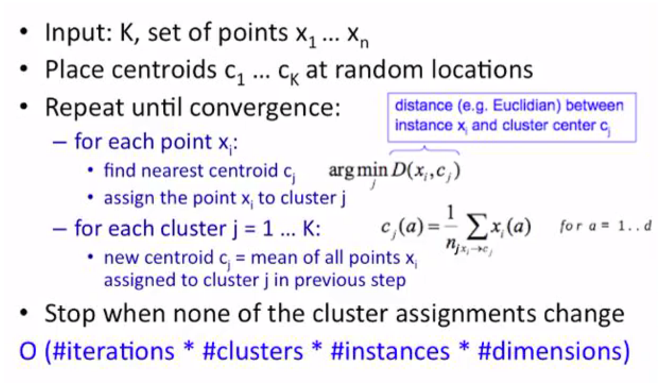


Code:

```
fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(cluster1[:,0], cluster1[:,1], s=30, color='r', label='Cluster 1')
ax.scatter(cluster2[:,0], cluster2[:,1], s=30, color='g', label='Cluster 2')
ax.scatter(cluster3[:,0], cluster3[:,1], s=30, color='b', label='Cluster 3')
ax.legend()
```

Summary

Algorithm: Summary



- Input: K, set of points $x_1 \dots x_n$
- Place centroids $c_1 \dots c_K$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j $\quad \underbrace{\arg\min_j D(x_i, c_j)}$ distance (e.g. Euclidian) between instance x_i and cluster center c_j
    - assign the point x_i to cluster j
  - for each cluster j = 1 ... K: $\quad c_j(a) = \frac{1}{n_j} \sum_{x_i \rightarrow c_j} x_i(a) \quad$ for $a = 1..d$
    - new centroid c_j = mean of all points x_i assigned to cluster j in previous step
- Stop when none of the cluster assignments change

O (#iterations * #clusters * #instances * #dimensions)

(Note: Only numeric attributes can be averaged for centroid calculations)

(Reference: K Means Clustering - Victor Lavrenko)

K-means

- Advantages
  - Scales well
  - Efficient
- Disadvantages
  - Choosing the wrong k

K-means

When to use?

- Normally distributed data
- Large number of samples
- Not too many clusters
- Distance can be measured in a linear fashion

Applications

- Business
  - Businesses collect large amounts of information on current and potential customers.
  - Clustering to segment customers into a small number of groups, for additional analysis and marketing activities
- Clustering for Utility
  - Efficiently Finding Nearest Neighbors
  - Alternative to computing the pairwise distance between all points

K-Means with Scikit-Learn

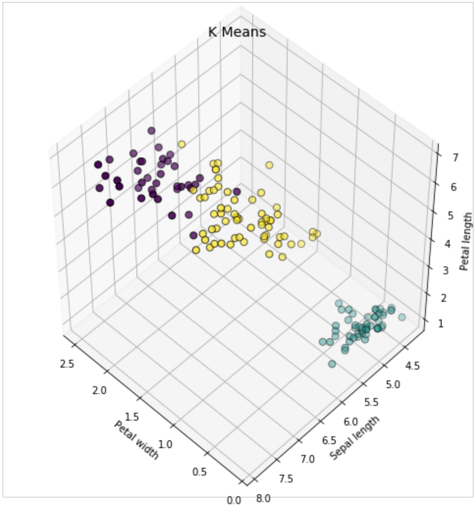
K-Means

```python
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# load the iris datasets
dataset = datasets.load_iris()
X = iris.data
# fit a K-means to the data
km = KMeans(n_clusters=3)
km.fit(X)
km.predict(X)
labels = km.labels_
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)
```

(Ref: Clustering Based Unsupervised Learning - Syed Sadat Nazrul)

K-Means Plotting



(Ref: Clustering Based Unsupervised Learning - Syed Sadat Nazrul)

Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com