clustering

k-means

finding k points as mean of k groups of data

k-modes

what if data is non-numerical? k-means rely on math calc (mean, euclidian distance) k-modes = extension of k-means

use dissimilarities (total mismatch between two objects) and modes mode = vector of elements minimize dissimilarities

k-prototypes

combine k-means and k-modes for numerical and categorical data

guassian mixture model

conventional : single gaussian mixture: can only use 1 average value vector to represent many values, effect not good vector quantization: use multiple important position to represent whole vector space, but not showing distribution size & shape

guassian mixture: use multiple guassian distribution, better representation

DBSCAN (Density-based spatial clustering of applications with noise)

definitions : core point = at least m points within distance e non-core point = < m points within distance e, but reachable from core points outlier = not reachable from any other point

connectedness = the extend of clusters found by DBSCAN cluster has 2 properties:

1. all points within cluster mutually density-connected
2. point is density-reachable => part of cluster

adv

- can find non-linearly separable clusters, cannot be adequately clustered with k-means / Guassian Mixture EM clustering
- find arbitrary shaped cluster
- robust to outliers

hierarchical clustering

combine or divide dataset into clusters iteratively => tree-like hierarchical structure created

- agglomerative (bottom-up, start from tree leaves, combine 2 nearest clusters)
- divisive (top-down, start from root of tree and select cluster to split at each iteration)

Dimension reduction

Principal component analysis

orthogonal transformation that convert set of observation of possibly correlated variables -> set of linearly uncorrelated variables (principal components)

(n observations, p variables) => #principal component = min(n-1, p) 1st principal component(PC) = largest variance (getting most variability) 2nd PC = highest variance possible under 1st PC ... => orthogonal basis set

usage:

- visualize genetic distance and relatedness between populations
- simplest true eigenvector-based multivariate analyses
- get lower-dimensional picture, projection with most informative viewpoint

Singular value decomposition

M = m x n real/complex matrix U = m x r real/complex unitary matrix, where U*U* = *UU* =I E = r x r rectangular diagonal matrix, non-negative real num on diagonal, sort in descending order V = n x r real/complex unitary matrix

M = U E V*

r suppose to be a smaller number eg. U = (m documents, r concepts) E = rxr (strength of each concept) [r = rank of matrix M] V = (n terms, r concepts)

M = e1*u1*v1^T + e2*u2*v2^T + ... (ei = scalar)

- always possible to decompose M into unique U,E,V for M = User-to-Movies matrix, U = user-to-concept similarity matrix E = strength of XX-concept V = movie-to-concept similarity matrix

theory

when dimensionality of data increases, volume of space increase quicker => data become more and more sparse

to do statistical modelling, need increse data points covariance = measure of joint variability of two random variable, greater value, more similar behavior covariance matrix = matrix of convariance of elements in i,j position of random vector

PCA algorithm:

1. subtract mean, scale dimensions by variance
2. compute covariance matrix

covariance matrix is expensive to compute when dimension is large => use SVD instead for E = r x r, reduce dimension number from m -> r, then obtain r-dimensional hyper-plane, value of E gives you amount of variance retained by this reduction

- eg. 1st PC get 37% of variance, next quickly drops

- better to find PCs that explain 80-90% variance of data

eg. reduce 64x64px image (4096 dimensions) to 50 engenvectors (50 D)

Linear Discriminant Analysis

extension of PCA(unsupervised), kind of supervised learning aim: want between-class scatter as large as possible, within-class scatter as small as possible better performance than PCA

Fisher criterion = J(W) = S_between * (S_within)^-1 solution = max (w^T*S_b*w)

latent Dirichlet allocation

allow set of observations to be explained by unobserved groups that explain why some parts of data are similar

assume words generated by topic

- document generation process: corpus-level(alpha) > document-level(theta) > word-level (z -> w) beta[corpus-level] -> w

early time document model

TF-IDF

TF: find how frequent keywords exist in document IDF: how many document contains this keyword

assume there's N documents, M words => build M*N matrix row = 1 document, column = tf-idf value of word in this document

disadv: fail to compress document info a lot, fail to extract info between words and documents

LSI

SVD on tf-idf matrix (U(Mxr)*E(rxr)*V(rxN)), r << N r = #topic, U = word-topic relation,
E = topic itself, V = topic-document relation

adv: compress word list a lot, better discover synonym

Unigram

word extract from single multimodal distribution

mixture of unigram

introduce hidden variable Z, chosen Z, p(w|z) generate N words

pLSI

1 document can have multiple topic, get weighted average of multiple topics disadv: poor generalization, fail to test on document with too many unknown words

- bag-of-words assumption

1. all N words' order doesn't matter
2. document order not matter

Classificaion

Naive Bayes

simple probabilistic classifier based on Bayes' theorem with strong independence assumption between features

Bayes' theorem: how much you can trust evidence assume all variables independent

pros: very fast, can do real time analysis; perform better on independent variables

cons: real world not likely independent

decision tree

Information_Gain = Original_info - left_frac*left_info - right_frac*right_info

measure of info:

1. Entropy I(t) = -sum{ p(i|t)*log_2[p(i|t)] }
2. Gini Impurity I(t) = sum{ p(i|t)[1 - p(i|t)] } = 1 - sum{ p(i|t)^2 }

Random Forest

collection of decision tree Ensemble learning: combine multiple weaker model to form robust model VS some other algorithm (eg. NN) aim to produce single strong model

after combine several models, should be better than any smaller model

- not easy to have overfit algorithm:

1. N = training sample, M = #feature
2. input feature number m << M
3. draw n samples out of N to form training set, use non-selected data for testing, evaluate error
4. for every node, random select m features, decide best partition method
5. every tree will not prune

AdaBoost

1. init x1...xn , yn samples, max iteration num
2. use weak classifier, get error
3. update weighting distribution

Regression

analyse relationship between dependent and independent variables find function that can represent points as much as possible

estimation method

method of moment ordinary least square estimation maximum likelihood estimation

model

unknown parameter (b), dependent variable (Y), independent variable (X) Y ~ f(X,b)

types

linear

simple linear regression multiple regression analysis log-linear model

non-linear

logistic regression partial regression

time series data, only Y data:

- autoregressive model
- autoregressive moving average model
- autoregressive integrated moving average model
- vector autoregression model

bagging (bootstrap aggregating)

performance of different ML algorithm

depends on size, structure of data

linear and polynomial regression

fast to model, good for not extremely complex relation, not a lot of data simple to understand challenging for non-linear data

neural network

have many layers, effective to model highly complex data flexible structure learning any kind of relationship giving more training data, better performance not easy to interpret and understood computationally intensive, hyper-parameter tuning require lot of data

regression tree

tree induction is task of taking set of training instance as input, decide which attributes best to split on, splitting dataset, recurring on resulting split dataset

- purity: information gain

random forest

can be classification & regression

randome forest = ensemble of decision trees

1. great at complexity, highly non-liear relationship, high performance on par with NN
2. easy to interpret and understand, decision boundaries built easy to understand
3. prone to major overfitting, model can be overly complex, contain unnecessary structure [use proper tree pruning, larger random forest ensembles to alleviate]
4. slower, require more memory for larger random forest

Boosting

train several smaller models to make bigger one

Ada boost

parameterization simple quite robust to overfitting perform well for large amount of data time consuming, take lot of memory

SVM