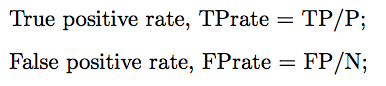
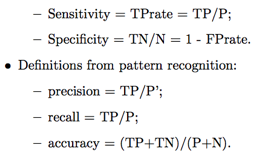
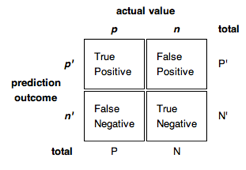
**Evaluation**

true positive = guess

the above is a confusion matrix

**sensitivity –** TP/(TP+FN) also called recall

**specificity –** TN/(TN+FP)

**precision –** TP/(TP+FP)

high sensitivity means few FN, low many FN

ROC- receiver operating characteristic

ROC space – precision/recall curve

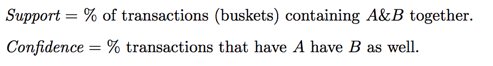
Area under ROC curve roughly equal to the probability of classifying a positive same as positive (TP rate)

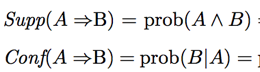
ROC curve plots TP rate (sensitivity) on the y-axis and FP rate on the x-axis

To make ROC curve first you predict your data then you compute the FP/TP rate using predicted labels comparted to actual labels, connect all the dots

Accuracy of A can be estimated as the area under the curve (if curve is diagonal accuracy of 50%)

**Association Rules** – want to identify correlations btw data (doe­­s A🡪B)





**support** – do they appear together

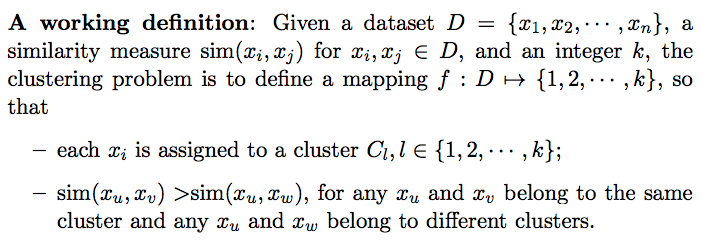
**confidence** – if one appears does it imply the other

**Anti-monotonic property** – if an item set is not frequent all of its supersets cannot be frequent OR if an item set is frequent all of its subsets cannot be frequent

**Apriori Algorithm**

* Find all frequent itemsets (above min support level)
* Join step (if OP, OM are frequent check OPM most likely to be frequent)

Order matters in AR mining 1🡪3 != 3🡪1

**Clustering**

**Distance between clusters**

* **Single link** – distance btw closest elements in cluster
* **Complete link** – distance btw furthest elements in cluster
* **Average link** – average across all pairwise combinations of elements in cluster
* **Centroid** – distance btw centers of clusters

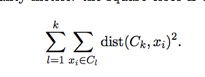
**Agglomerative clustering**

* Start with each data point as a cluster
* Repeatedly combine two closest clusters into one
* Until form one cluster
* Time O(kn^2) k num clusters
* Space O(n^2)

**Divisive clustering**

* Start with whole dataset as one cluster
* Repeatedly split until each cluster has one data point
* Space – O(n^2)

**Partitioning clustering**

* Create clusters in one step
* Need to know num clusters (k)****square error metric

**K Means**

* Randomly pick k objects from D as representatives of k clusters
* Update class memberships – assign each object to its closest class
* Recompute the k centroid (use average of all values of k)
* Repeat until no object changes its class membership

**EM Algorithm**

Mixture models, each cluster corresponds to a probability

For EM algorithm want to find parameters (mean, covariance)

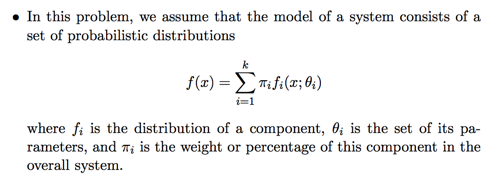
Have a certain number of points (suspect they came from k-different Gaussian distributions)

* Start by placing k random Gaussians
* for each point P(B|x\_i) calculate the probability that point came from distribution B
* Once you have computed assignments use those numbers to reestimate mean and variances
* Iterate until convergence

E-step: perform probabilistic assignments of each data point to some class based on the current hypothesis *h* for the distributional class parameters;

M-step: update the hypothesis *h* for the distributional class parameters based on the new data assignments.

**Probabilistic model-based clustering**



want to estimate model parameters and component weights, which can be viewed as class memberships

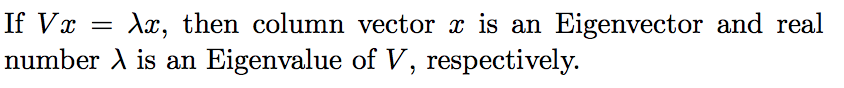
**Principal Component Analysis**

Want to map data to a lower dimensional space, where patterns in the data emerge and composite features can be identified

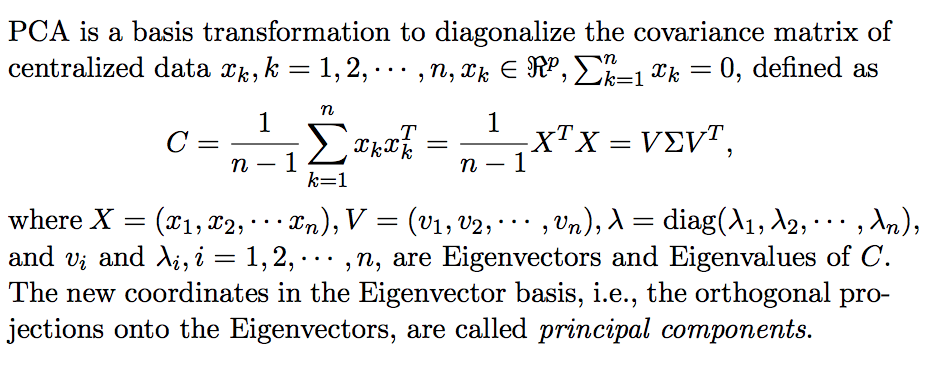
* Adjust (linearly) the view point so that the direction of the major variance becomes one of the coordinates
* PC’s are coordinates along which the data vary the most
* Compress data by taking fewer coordinates (less data but may lose info)

Want to describe original data in a new basis (eigenvector space)

Do this by finding the eigenvectors and eigenvalues of a transformation matrix



Lambda is the “stretching factor” here



Theorem: A symmetric matrix is diagonalized by a matrix of its orthonormal eigenvectors

Covariance matrix is always symmetric

**PCA Procedure**

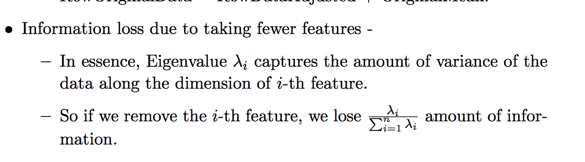
Subtract the mean of each feature (dimension) must 0 center

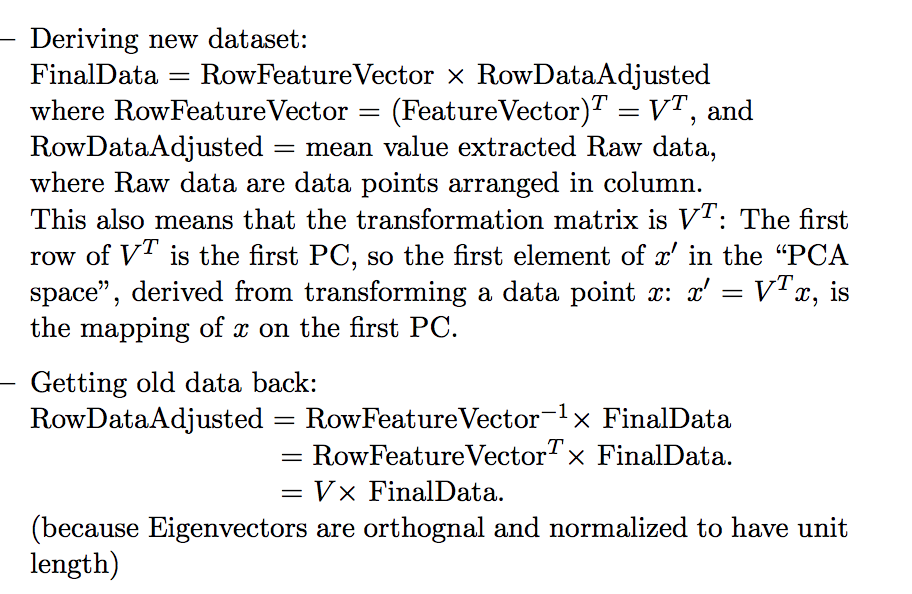
Compute the covariance matrix

Compute eigenvectors/eigenvalues of the covariance matrix C

Create feature vector made of m eigenvectors

Here begin finding projections of each data point to transform it to a lower dim





PCA achieves

* Dimension reduction: finds an embedding of data in lower dimensional space to reduce storage/comm cost
* Remove possible noise (could remove impt info)
* Matrix factorization: general paradigm for data analysis

Data compression and noise removal, factor analysis, anamoly

and outlier detection, data viz

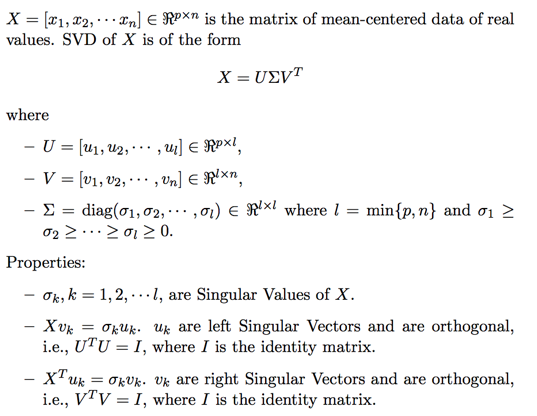
**PCA Fails –** covariance extremely sensitive to large values

If each dimension is not normalized to zero mean and unit variance

Assumes the underlying subspace is linear (if it data is low dimensional is non linear PCA will fail)

**Singular Value Decomposition**

PCA can only be performed on squared covariance matrix, SVD can be applied directly to the input data matrix (more general than PCA)



PCA and SVD are both linear transforms/matrix factorization

Whenever we do PCA we can do SVD as well

SVD carries out 2 PCA’s simultaneously one on the individual data points and one on the individual features

* The left singular vectors U in SVD are the same as the eigenvectors of the covariance matrix on features (C= DDT)
* The right singular vectors V in SVD are the same as the eigenvectors of the covariance matrix on data points (DTD)

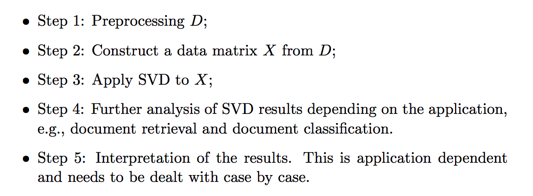
Both the left and right singular vectors form orthonormal bases

**Latent Semantic Analysis**

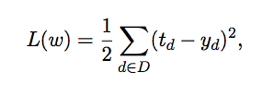
An application of SVD for text document analysis (text mining)

Most important words/phrases for a set of documents

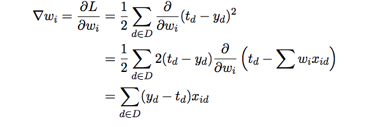
**Words of similar meaning occur often in similar documents**

****

**Neural Networks**

Loss function that we will use for NN

Want to minimize the error using gradient descent

General equation to reduce the error – following gradient descent

Can use SGD which instead randomly selects one data points and moves in the direction of the gradient on this data point instead of over all data points

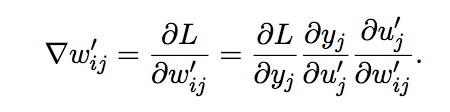
update to the weights in our weight vector

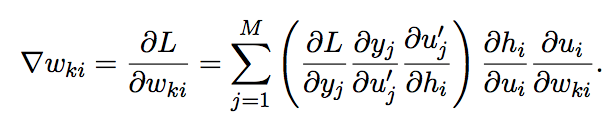
**Stochastic Gradient Descent** – select a random point from your dataset and backpropogate the error from that data point through your neural network vs

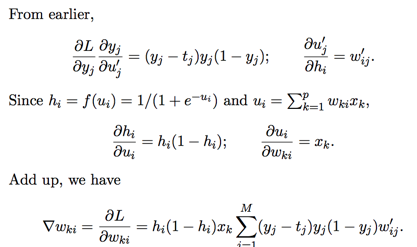
**Gradient Descent** – take all data points calculate loss (error) over all data points and backprop this error through the system

**Backpropogation**

* Initialize the network – start with random initial values of the model parameters (w)
* Push an input from the input to the output – compute the error for this example
* Back propogate the error at the output layer into network to compute the gradient

adjustments to weights at the hidden layer to output layer

adjustments to weights at input layer to hidden layer

if using sigmod as activation

* Alternate among examples (SGD)

In general neural networks are *fully connected* meaning that each node is connected to all the nodes in the next layer

**Deep Learning**

A deep neural network is a network that has multiple layers, perceptron’s with special functions or crazy network architecture

* Specialized programs tailored to specific problems

**Autoencoder –** special version of NN (trying to recreate inputs at the ouputs)

* Transform data to compressed, lower dimensional space then reconstruct the data from compressed version
* Has a symmetric structure (same number of input and output nodes)
* Use backprop to train autoencoder

Can be used for

* Data compression/pattern identification
* Pretraining a NN: find intrinsic features in the data that are potentially easier to find local minima through

**Convolutional Neural Networks**

Used for topographical data (data that is not the same if you swap the rows and columns)

Classification, localization (objects of interest in image), detection (localization of all objects in an image), segmentation (give a class label to every object in an image), outline (outline the contour of all objects of interest and label)

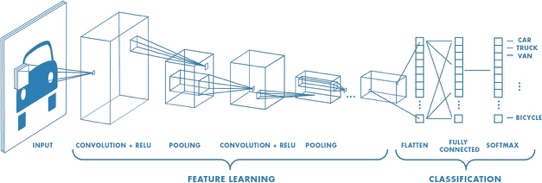
**Convolution** – way to combine matrices (measures similarity between two matrices sort of)

We have a sliding frame that we slide over the image to find different features/edge patterns in the image

Result of this operation is an activation map (shows where certain features can be identified in the image)

Initially the filter has random weights, at the end of training its weights capture features in the data

Pooling deals with image invariants (different positions)

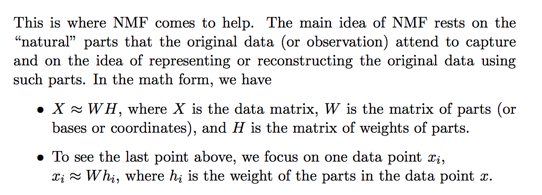


**ReLU –** non-linear activation function 

Helps to solve the vanishing gradient problem (where the gradient gets so small it is not actually detecting the weights)

Vanishing gradient problem – as we back proprogate the gradient (error signal) decreases exponentially with n while the front layers train very slowly

**NNMF**



**PCA Proof**

