GENERAL

$\log_a(X^P) = plog_a(x)$
Cross-entropy/Logistic Loss = - $\frac{1}{N}{}_{n=1}^{N}[y_n log\hat{y}_n + (1-y_n)log(1-\hat{y}_n]$
Cross-entropy = - log likelihood. A way of comparing distributions. KL Divergence is a way of comparing probability distributions.
Normal equations can be derived from minimizing empirical risk, assuming that P(y|X=x) is normally distributed with mean $w^T x$ and variance σ^2, assuming that $y = w^T x + \epsilon,$ where $\epsilon \approx N(0, \sigma^2)$
Isocontours of Gaussian distribution have axes whose lengths are proportional to the square root of eigenvalues of the covariance matrix.

Bayes Rule = $P(X,Y) = \frac{P(X)P(Y|X)}{P(Y)} = \frac{P(Y)P(X|Y)}{P(X)} = \frac{P(B|A)P(A)}{(P(B|A)P(A)+P(B|)P()}$
Bayes Optimal Discriminant Classifier/Optimal Decision Boudary for unknown P(X,Y) = f*(x) = P(Y=1|X=x) = P(Y=-1|X=x)
Naive Bayes Classifier: argmax$_y \prod P(X = x_i|Y = y_i)P(Y = y)$
Bayesian learning: one seeks to estimate for a new example x the probability P(y|x,D) by integrating over all possible choices of f, using P(f|D): P(y|x,D) = $\int P(y|x, f)dP(f|D)$.

VC Dimension: capacity (complexity) of a model. C = training examples that can be seperated, regardless of label assignment. For linear, C = d

---

Leave One Out Cross validation: By repeating this we can get a bound on our error rate. Virtual leave one out is a derivation of a learning method to do this at train time so that we get this error bound for free.

PSD = $x^T Ax \geq 0$; Symmetric PSD matrix != all non-negative elements; PSD diagonals = non-negative

---

Eigenvectors/Values = Av = $\lambda v, (A - \lambda I)v = 0$

Independence = P(X,Y) = P(X)P(Y), C(x,y) = (x-$\mu_x).(y - \mu_y)/(sxsy)$

If X1 and X2 are normal + ind. (X1,X2) must have mv normal dist. => true. If (X1,X2) = mv normal dist, not independent.

MATRIX IDENTIES

$(A^{-1})^T = (A^T)^{-1}$	$(AB)^{-1} = B^{-1}A^{-1}$
TR(A) = $\Sigma eigenvalues$	|A| = eigenvalues
$\frac{\partial TR(A)}{\partial A} = I$	$\frac{\partial TR(XA)}{\partial X} = \frac{\partial TR(AX)}{\partial X} = A^T$
$\frac{\partial TR(X^T AX}{\partial X} = (A + A^T)X$	$\frac{\partial ln\|X\|}{\partial X} = X^{-T}$
$\frac{\partial a^T x}{\partial x} = \frac{\partial x^T a}{\partial x} = a$	$\frac{\partial x^T Ax}{\partial x} = (A + A^T)x$
$\frac{\partial a^T Xb}{\partial X} = ab^T$	$\frac{\partial a^T X^T b}{\partial X} = ba^T$
$\frac{\partial a^T Xa}{\partial X} = \frac{\partial a^T X^T a}{\partial X} = aa^T$	$\frac{\partial a^T X^T CXb}{\partial X} = C^T Xab^T + CXba^T$

$\frac{\partial |X^T AX|}{\partial X} = |X^T AX|(AX(X^T AX)^{-1} + A^T X(X^T A^T X)^{-1})$
$\frac{\partial((Xa+b)^T C(Xa+b))}{\partial X} = (C + C^T)(Xa + b)a^T$

LINEAR REGRESSION

Assumes no multicollinearity, homoscedacity, normally distributed data

Risk	Loss	Gradient
$\Sigma_k l_k(x^k, y^k)$	$(x^k w - y^k)^2$	$2(X^T Xw^T - X^T y)$

w = $(X^T X)^{-1} X^T y$

Ridge regression = MAP estimate with a gaussian prior.
Lasso Regression = MAP estimate with a laplace prior.
Pseudo-Inv: $(X^T X + \lambda I)^{-1} X^T y = X^T (XX^T + \lambda I)y$

LOGISTIC REGRESSION

Soft + flexible + modular decisions, probabilistic output, change decision boundary manually. We model the decision boundary as f(x). Logistic function S is a mapping from [-inf, inf] $\rightarrow$ [0, 1].
In logistic regression, only the marginal examples significantly contribute to determining the position and slope of the sigmoid. LR can be motivated from a generative model with Gaussian or poisson class conditionals. Therefore we can solve logistic regression via MLE.

Risk	Loss	Gradient
	$P(Y = 1\|X = x) = \frac{1}{1+e^{-f(x)}}$	

Logistic Function: S(t) = $g^{-1}(t) = \frac{1}{1+e^{-z}} = S(f(x)) = P_f(Y = 1|X = x)$
[non]Linear Logistic regression (log odds ratio/logit): log($\frac{P_f(Y=1|X=x)}{P_f(Y=-1|X=x)}) = f(x)$
Kernelized Logistic Regression: $\Delta w \approx s(-z_k)y_k\phi(x^k), \Delta\alpha_k\ S(-z_k)y_k$

PERCEPTRONS/SVM

Given linearly seperable data, the perceptron algorithm will take no more than R^2/γ^2updates to converge. Where$R = max_i||x||_i$is the radius of the data and $\gamma = min_i \frac{y_i(\alpha\cdot x_i)}{||\alpha||}$
Risk = R[w] = $\sum_k l(x^k, y_k)$, Loss $= max(0, -z), f(x) = sign(\Sigma_k\alpha_i\phi(x_k))$
Hard Margin: m = 1 / ||w||
Optimum margin: $\Delta w_i = \eta yx_i$ if incorrect else 0
Optimum margin: $\Delta\alpha^k = \eta y^k x_k$ if incorrect else 0
Geometric(1/||m|| and functional margin link: w $\cdot x_1 + b = 1, w \cdot x_2 + b = -1$
w $(x_1 - x_2) = 2$
Dist $x_1, x_2 = 2m = ||x_1 - x_2|| = (x_1 - x_2) \cdot w/||w|| = \frac{2}{||w||}$

No matter the dimensions, the minimum number of required support vectors is 2. Good fit = allow a few training errors, good robustness = maximize the margin for a classifier. Slackness = every training point misclassified by a soft-margin SVM is support vector. We need only the dot product of $x_i, x_j\forall i, j$. and no more info. Strong duality holds for HM and SM SVM.
Soft Margin = min $\frac{1}{|m|} + C * R_{train}$, Hard Margin $= minC * R_{train},$
Soft Margin SVM = as C $\rightarrow$ 0, width of margin $\frac{2}{||w||} \rightarrow$ inf
Large C = focus on a fit to the data, small margin is OK[hard margin]. Small C = focus on large margin, less tendency to overfit.

GRADIENT DESCENT

We can converge to global optimums in gradient descent [variety of versions] if we are working within a convex loss function. They are convex in a NN if we don't have hidden layers.

GD: $\partial Risk/\partial w$	SGD: $\partial Loss/\partial w$
$\Delta w_{gd} = -\eta\nabla_w R - \gamma w$	$\Delta w_{sgd} = -\eta\nabla_w L - \gamma w$
$w_{t+1} = w_t - \gamma(1/n)\Sigma_n\nabla_w l(f_w(x^k), y)$	$w_{t+1} = w_t - \gamma_t\nabla_w l(f_w(x^k), y)$

Regression Example
R[w] = $\Lambda(Xw - Y)^2 = (Xw - Y)^T\Lambda(Xw - Y) = w'X'\Lambda XW - 2y'\Lambda Xw - y'\Lambda y$
$0 = \frac{\partial w'X'\Lambda Xw}{\partial w} - 2\frac{\partial y'\Lambda Xw}{\partial w} - \frac{\partial y'\Lambda y}{\partial w}, \beta = X'\Lambda X, \alpha = y'\Lambda X$
$0 = \frac{\partial w'\beta w}{\partial w} - 2\frac{\partial\alpha w}{\partial w} = (\beta + \beta^T)w - 2\alpha^T = 2X'\Lambda Xw - 2X'\Lambda y$
w = (X' $\Lambda X)^{-1}X'\Lambda y$
Given, R[w] = $\Lambda(Xw - y)^2 + \gamma w'w[regularized]$
$0 = 2X'\ \Lambda Xw - 2X'\Lambda y + \frac{\partial\gamma w'w}{\partial w} = 2X'\Lambda Xw - 2X'\Lambda y + 2\gamma w$
w = (X' $\Lambda X + \gamma I)^{-1}X'\Lambda y$
With Weight decay: $\alpha_h^{(t+1)} = \alpha_h^{(t)} - \eta\gamma\alpha_h^{(t)}$ (For other examples $h \in \{1, 2\ldots, m\}/i)$

Loss Functions:
Losses as an expression of a function of the functional margin (z), z = y f(x)

Zero-One	L(z) = 1(Sign(f(x)) $\neq y) = 1(z) \leq 0)$
Square	L(z) = (f(x) - y)$^2 = (1 - z)^2, y = +/-1$
Hinge	max(0, 1-z) (svc loss)
Logistic	log(1 + e^{-z})
Perceptron	max(0, -z)

KERNELS

Kernel = similarity measure, a dot product in some feature space. What makes a good kernel? Symmetric, Kernel matrix K is invertible (satisfies Mercer's condition), Kernel matrix is PSD if eigenvalues are positive or if it is an outer product.
Classifier: f(x) ${}_k\alpha^k k(x^k, x) + b$
Gaussian Kernel is a subset of RBF kernel
Gaussian: k(s, t) = $e^{-||s-t||^2/\sigma^2}$, Polynomial:$k(s, t) = (s.t)^q$
$([s_1, s_2] \cdot [t_1, t_2])^2 = [s_1^2, s_2^2, \sqrt{2}s_1, s_2] \cdot [t_1^2, t_2^2, \sqrt{2}t_1 t_2]$
K(x,y) = $\left(\sum_{i=1}^n x_i y_i + c\right)^2 =$
$\sum_{i=1}^n (x_i^2)(y_i^2) + \sum_{i=2}^n\sum_{j=1}^{i-1}\left(\sqrt{2}x_i x_j\right)\left(\sqrt{2}y_i y_j\right) + \sum_{i=1}^n\left(\sqrt{2c}x_i\right)\left(\sqrt{2c}y_i\right) + c^2$

---

kernel Machines + Dual Representation (parametric, non-parametric)
f(x) = w $\phi(x) = \Sigma_k\alpha_k k(x^k, x), w = \Sigma_k\alpha_k\phi(x^k), k(x^k x) = \phi(x^k)\phi(x)$

Parzen windows = assign x to the class label of the majority of the examples enclosed in a sphere of radius σ, $f(x) = \Sigma_k y_k k(x, x_k)$.
Parzen window for the linear kernel is just Hebb's rule, k(x, x_k) = $x.x_k$

MLE & MAP

MLE = a point estimate, not a dist. Maximize probability given the model, = P(X|θ) = $\Sigma log(P(X|\theta))$
MLE Steps = get Log likelihood. derive. set to 0. solve.
Expected Value steps = Get CDF, multiply by values. Now that we have those values we can go about calculating the expectation which is simply the probabilities of each of those happening multiplied by the point values in order to get the expected value.
if f(x;θ) = PMF, then $l(f) < 1$

Maximum Likelihood for exponential distribution.
P($x_i|\theta) = \theta e^{-\theta x}$
lik(θ) = $\prod^n p(x_i|\theta) = \sum_{i=1}^n log(p(x_i|\theta)) = \sum_{i=1}^n log(\theta) - \theta x_i = nlog(\theta) - \sum_{i=1}^n\theta x_i$
max. lik(θ) = $\partial y/\partial\theta = \frac{\partial nlog(\theta) - \sum_{i=1}^n\theta x_i}{\partial\theta} = 0$
$n/\theta - \sum_{i=1}^n x_i = 0, \theta = \frac{\sum_n x_i}{n}$

Maximum a posteriori: Maximize the model given the data. Use bayes rule to invert the ML formula. The MAP estimate allows us to inject into the estimation calculation our prior beliefs regarding the parameters values in Θ.
= argmax(P(model|data)) = argmax(P(data|model)P(model)) / P(data) (P(data) is constant, so we remove it)
= argmax(P(model|data)) = argmax(P(data|model)P(model)) = argmin -log(p(d|m)) - log(p(m)) = - log likelihood - log prior

---

Prob. of Linear Regression: P(y|x,σ^2) $\approx N(w^T x, \sigma^2)$ with prior $p(w)$.
Assume Laplace, show equivalent to minimizing R(w) = $\Sigma_n(y_k - w^T x^k)^2 + \lambda||w||_1$
$w_j \approx Laplace(0, t), P(w_j) = \frac{1}{2t}e^{-|w_j|/t}, P(w) = P(w_j) = (\frac{1}{2t})^D \cdot e^{-(\Sigma|w_j|)/t}$
P(w|$X_i, Y_i) \propto (\prod N(Y_i|w^T X_i, \sigma^2)) \cdot P(w)$
l(w) = $\Sigma log(N(""")) + \Sigma logP(w_j)$
= - $\Sigma\frac{Y_i - w^T X_i)^2}{2\sigma^2} + \frac{-\Sigma|w_j|}{t} + nlog\frac{1}{\sqrt{2\pi\sigma^2}} + Dlog(1/(2t))$
First two terms are our only w terms, last two are constants.
= $\Sigma(y_i - w^T x_i)^2 + 2\sigma^2/t\Sigma|w_j|$
The last term becomes the lambda in front, proving that it's equal to the above Risk function.

GAUSSIAN CLASSIFIERS/LDA/MIXTURE MODELS

Gaussian Classifier = P(X=x | Y=y) $\propto e^{||X-\mu^{class,y}||^2/2\sigma^2}$
guassian: P($x_i|\mu_i, \sigma_i) = \frac{1}{(\sigma_i\sqrt{2\pi\sigma})}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Log likelihood of guassian: l($X_1, ...X_N; \mu, \sigma) = Nln(\frac{1}{\sqrt{2\pi\sigma}}) - \frac{1}{2\sigma^2}\sum(X_i - \mu)^2$
$\mu = 1/N\sum X_i, \sigma = \sqrt{(1/N)\sum(X_i - \mu)^2}$
Centroid method = f(x) = $(\mu_1 - \mu_0) \cdot x + b = 0$
Isotropic Gaussian model: patterns x are generated from a template (class centroid) plus some gaussian noise with 0 mean and same variance. Shrunken centroid method takes all this a bit father by rescaling and selecting the most informative features.
Isotropic Gaussians (Same var, in all directions): f(x) = $(\mu^1 - \mu^0) \cdot x + b, b = (\mu^{0,2} - \mu^{1,2})/2 + log(N_1/N_0)$
Correct if different scales of features(Sphering): f(x) = $(\mu^1/\sigma - \mu^0/\sigma) \cdot x + b, b = ((\mu/\sigma^2)^{0,2} - (\mu/\sigma^2)^{1,2})/2 + log(N_1/N_0)$
Most general case(LDA, this is equivalent to whitening): f(x) = $\Sigma^{-1}(\mu^1 - \mu^0) \cdot x + b, b = ((\mu^0\Sigma^{-1}\mu^0 - \mu^1\Sigma^{-1}\mu^1)/2 + log(N_1/N_0)$
LDA is a generalization of the Gaussian classifer for cases in which the input variables are not statistically independent, but all classes have the same covariance matrix. Once we rotate the input space into the principal axes of the covariance matrix and rescale by the eigen values, LDA is like the centroid method.
P(X=x|Y=y) $\propto exp(-1/2(x - \mu^y)\Sigma^-1(x - \mu^y)^T$

When we do not hold the covariance of the classes as constant, we get QDA.
Maximize $\frac{\mu_1^2-\mu_0^2}{\sigma^2}$, new $f(x) = \Sigma^{-1}(\mu_1^2 - \mu_0^2) \cdot x + b$
Pooled within class Covariance Matrix = Σ_{LDA}
When we use shrinkage with LDA and have balanced binary classes of -1 and 1, it can be shown that the decision boundary created is exactly equal to the ridge regression of the data. Except for differing covariance matrices, where ridge regression used whole covariance.

Mixture Models: Mixed clusters of data. Some smaller, some larger for a given class.
f(x) = $\sum_k \alpha_k k(\mu^k, x; \sigma)$
P(X=x|y) = $\Sigma_k P(X = x, S = s_k|Y = y)P(Y = y)$
$= \Sigma_k P(X = x, S = s_k|Y = y)P(S = s_k|Y = y)P(Y = y)$
$\Sigma_k P(X = x, S = s_k|Y = y) \propto exp(-||x - \mu_k||^2/2\sigma^2)$
P(S=s$_k|Y = y)P(Y = y) \propto \alpha_k$

GENERATIVE AND DISCRIMINATIVE

Generative = model P(Y) P(X|Y) as prior and maximum likelihood/maximum a posteriori. Use those to generate P(Y|X).
Discriminitive = model P(Y|X) directly $\rightarrow P(X, Y) = P(Y|X)P(X)$
A discriminant function f(x) is a function such that f(x) > 0 for 1 class and f(x) < 0 for the other. f(x) = 0 is the equation of the decision boundary. Given w, f(x) = wx is a linear discriminant function. Corresponding decision boundary w.x=0 is a hyperplane (a subspace of dimension (d-1)). We can transform x into another space to get non-linear decision boundaries.

Risk, Loss & SRM

Risk = Sum of the losses, Risk function = convex if Hessian = PSD
Reduce risk + understand error = (1)Nested subsets of models (2) shrinkage (3) nested subsets of kernels
Empirical Risk is the average loss over a finite number of given examples, Empirical Risk: r$_{train}[f] = (1/N)\Sigma_k L(f(x^k), y)$
Expected Risk: The expected value of the loss, i.e. the average over an infinite number of examples.
Expected Risk/Generalization Error: R[f] = $\int L(f(x, w), y)dP(x, y)$
The problem is, we don't know that P(x, y) we only get test examples to get Empirical Risk: R$_{test}[f] = (1/N)\Sigma_k L(f(x^k), y)$
Guaranteed Risk: Upper bound on the expected risk, measure of training risk + some pre-determined error bar that's a function of the complexity and number of examples. As training error decreases and model complexity increases, we are reaching the guaranteed risk.
R$_{gua}[f] = R_{train}[f] + (\delta, C/N), R[f] \leq r_{gua}[f]$
MAP is a method of SRM where -loglike = emp. risk, regularizer = negative log prior

SCORING AND EVALUATION

Error rate = E, Accuracy = A: A = 1 -E
AUC for ROC: y = Positive Class Success Rate, x = false alarms
Error Rate is binomial: f(k;n,p) = Pr$(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$
Error Bars, p + (1-p) = 1, binomially distributed. Expected = np, Variance = np(1-p). Expected value of error rate E = x= p
Error bar of error rate with n test examples, $\sigma = \sqrt{E(1-E)/n}$
Use the bootstrap rather than knowing the distribution of our favorite cost function. Resample with replacement.
Bonferroni Correction: New necessary p-value = n-trials * p-value

MODEL SEARCH

Lots of hyperparameters to tune, preprocessing, model, model hyperparameters, loss function, regularizer, learning rate and more.
Brute force / Grid Search: Simple, global minima, but scales poorly. Can do fancy versions like simulated annealing and random walks but these are fairly intensive/complicated methods. Greedy search chooses a random direction and walks until a minimum is reached then tries another value.

Filter Methods: Leverage knowledge in order to simplify the search base.

Wrapper Methods: your machine = black box and you only tune hyperparameters. ie CV.

Embedded Methods: push hyperparameters down to the model level. Wrappers are very computationally expensive and overfit so we try and reduce them down by pushing to the model level. We need two levels of inference to enjoy finite capacity of the learning problem. Optimizing the kernel parameters leads to infinite VC dimension (can learn perfectly any training set); optimizing the ridge or regularization parameters leads to zero capacity.

NEAREST NEIGHBORS

N $\rightarrow$ inf, error is at most only twice as bad as bayes optimal
[good] = non-linear, non-parametric
[good/bad] = decide on labels at run time
[bad] = super slow, missing out w/o looking at labels
K-D Tree, improves search for KNN to log(n)

CURSE OF DIMENSIONALITY

D > N. Solutions are $[\uparrow N][\downarrow D]$ or you can try and reason in a lower dimension with regulariation, dim. reduction, better distance metric. Data reg ↑ exp w/ ↑dim.
Dim. reduction = PCA[extrinsic], think about a swiss role.

DISTANCES

To be a distance satisfy: (1) dis(x, x) = 0; (2) x then dis(x,x) = 0; (3) dis(x,y) == dis(y,x) (4) dis(x,y) ≤ dis(x,z) + dis(y,z)
Minkowski distance = Disp(x,y) = $({}_{j=1}^{d}|x_j - y_j|^p)^{1/p} = ||x - y||_p$
0-norm = number of non-zero elements, ${}^{d}(x_j - y_j)^0$
If x,y are binary == hamming distance, if x,y are non-binary = edit distance
Mahalanobis distance m $\Sigma^{-1}, dis_m(x, y|\Sigma) = \sqrt{(x-y)^T\Sigma^{-1}(x-y)}$
Using covariance matrix decorrelates and normalizes features.

TREES

Decision boundary will always be axis aligned, greedy w/ no global guarantees. However a infinite depth binary decision tree can always achieve 100Goal = increase purity over time, can measure purity with entropy. Random forests reduce variance.
KD Tree = O(d log n), only works when d « n, inexact: may miss neighbours
locality-sensitive hashing = O(d H), H«n... number of hashes, inexact: may miss neighbours
Entropy/Impurity of Collection of Samples = H = $\Sigma - p_i log_2 p_i$
MinEnt = all same, H = 0, MaxEnt = 50/50 [even chance], H = 1
Information gain = ent.(parent) - avg. ent.(children)
Bagging = bootstrap aggregation, take a sample of the data and use that. Will be approx 63
Boosting = Specialized model for a subset of the data, you're weight classifiers. Avg all models and attach weights
Adaboost = assign higher weights to misclassified examples. Iterative.

NEURAL NETWORKS

Regularization = dropout, remove some nodes at a point in time. Hyperparameters = hidden layers, functions, hidden units, learning rate, iterations, loss function.
Softmax + Cross-Entropy = better classification
BACKPROP:
w$^2 \leftarrow w^2 - \eta * \frac{\partial L}{w^2}, w^1 \leftarrow w^1 - \eta * \frac{\partial L}{w^l}$
$\frac{\partial L}{w^2} = y^1 \frac{\partial L}{\partial z^2}^T = y^1(diag(g'(z^2 + b^2)) \cdot \frac{\partial L}{\partial y^2})^T$

$\frac{\partial L}{w^1} = y^0 \frac{\partial L}{\partial z^1}^T = y^0(diag(t'(z^1 + b^1)) \cdot \frac{\partial L}{\partial y^1})^T = y^0(diag(t'(z^1 + b^1)) \cdot w^2 \cdot \frac{\partial L}{\partial z^2})^T$
squared loss // could remove
$\frac{\partial L}{w^l} = y^0(diag(1-tanh^2(z^1+b^1)) \cdot w^2 \cdot (diag(g(z^2+b^2)(1-g(z^2+b^2))) \cdot (y^2-y)))^T$
$\frac{\partial L}{w^2} = y^1(diag(g(z^2 + b^2)(1 - g(z^2 + b^2)) \cdot (y^2 - y)))^T$

CNNS

Convolutional layer: h$_j^n = max(0,_{k=1} Kh_k^{n-1} * w_{kj}^n).It'sadotproductwithsomeother$
Convolutional layer params = Num filters x dim. of filter.
Pooling has no weights, just makes it smaller by taking the max feature (or another pool, but commonly max).
Can always do same of CNN with NN of same size - just too many params. Waste of resources. Convolving the first filter in the input gives the first slice of depth in output volume.

UNSUPERVISED LEARNING

clustering	dim. reduction	mode-seeking
partition into clusters	discover low-d manifolds/good feats	frequent patterns

Another perspective:

partitioning	hierarchical
just grouping	taxonomy, bottom $\uparrow ortop \downarrow$

Agglomerative Clustering = what can I merge? Divisive is what can I chop up?
Hierarchical clustering doesn't need k param

KMEANS

Objective = $\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$
where μ_i is the mean of points in S$_i$.
Coordinate ascent on j and j must be monotonically decreasing. C and μ are latent
Create random centroid
repeat until convergence:
assignment: S$_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \; \forall j, 1 \leq j \leq k\}$
where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.
new means: m$_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$
KMEDIODS is a variation on KMEANS that looks at the neighborhood, choose the point that is most close to the center. A medoid can be defined as the object of a cluster whose average dissimilarity to all the objects in the cluster is minimal. i.e. it is a most centrally located point in the cluster. It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

SVD/PCA/CF

Factor a matrix into 2+ low rank matrices with the hope that you get a meaningful structure in the data.
PCA = Direction of maximum variance. PCA = eigenvectors of covariance matrix with large eigenvalues. The new features are linear combinations (weighted sums) of the old ones. They are obtained by rotating the input space into the axes of the principal components of XTX: X->XU, where the columns of U are eigenvectors. This transform has the following properties: (1) the eigen directions corresponding to the largest eigenvalues explain best the variance in the data; (2) If we limit ourselves to the n* eigen directions corresponding to the top eigenvalues and rotate back into the original axes: XU -> XUUT, the reconstructed data XUUT are closest to the original data X in the least square sense. So we cut down on the number of features with as small as possible information loss.
For example in a document term matrix, we would have documents that are similar and words that are similar to one another (columns vs rows). A matrix that encodes the data becomes a latent structure. We can find topics that way.
PCA = (1) Subtract mean (2)?Scale (3) Cov(X) =$X^T X$, get the k largest eig.
SVD = approximation when using only some latent factors. Alternative to PCA w/ same eigen-structure
X = USV^T
X$^T X = US^2U^T, dim(u) = (d, r_rank), dim(s) = (r_rank, r_rank) = UDU^T$
XX$^T = VS^2V^T, dim(v) = (N, r_rank), dim(S) = (r_rank, r_rank) = VDV^T = CSU^TUSU^T$
U = eigenvectors, s = singular diagonalized singular values, s$^2 = eigenvalues$

Whitening/Sphering:
X is Centered, COV(X) = X$^T X = USSU^T$
X is not Centered, COV(X) = Σ = E $\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}])(\mathbf{X} - \mathrm{E}[\mathbf{X}])^{\mathrm{T}}\right]$
Valid COV(X) values = +- square root of diagonals multiplied
U and V are unitary rotational matrices, Scaling Matrix = $\Sigma, SVD(X) = U\Sigma V^T$
if Centered Data, $\Sigma = X^T X$, Centering: $x^k - \mu$
To rotate: XU, U from $\Sigma^1 = UDU^T$
Standardizing/ Sphering: (x$^k - \mu)/\sigma$, Whitening: $\sqrt{COV(X)^{-1}} = \Sigma^{-1/2}$
Whitened Space(and how to send data to it): $\phi = X\Sigma^{-1/2}$
Application to Ridge Regression - if we want to vary lambda a lot or try a lot of different lambdas, it's worth it to perform this rotation because it will make it much easier to both invert and add to the diagonals because we've got this diagonalized matrix. Makes it a bit easier to manipulate.