Midterm Review
Machine Learning for Cities

March 2018
Alex Shannon

Notes on conceptual topics in machine learning in preparation for a midterm exam; for more math and code-oriented examples, please refer to the labs and homework assignments at https://github.com/clapham13/Machine_Learning_for_Cities

1. A Few Basic Concepts

Generalization: the ability to perform a task in a situation which has never been encountered before

Machine Learning: study of systems that improve their performance with experience (typically by learning from data)

Data Mining: process of extracting useful information from massive quantities of data

Classification: prediction of a discrete value; goal is to maximize proportion of correct predictions; *discrete*

Regression: estimate of a numerical value; goal is to minimize the mean squared error; *continuous*

Detection: identifying relevant patterns in massive, complex datasets (e.g. anomalies, clusters, noise/errors, events, etc.)

Supervised Learning: you have input variables (X) and output variable (Y) and you use an algorithm to learn the mapping function from the input to the output $Y = f(x)$

Unsupervised Learning: all data is unlabeled and the algorithms learn to inherit structure from the input data

Semi-Supervised Learning: you have large amounts of input (X), but only some of the data is labeled (Y); a mixture of supervised and unsupervised methods can be used

Reinforcement Learning: Sequential actions with delayed rewards; goal is to learn optimal action in each state

2. Decision Trees

A *decision tree* is a set of rules that can be learned from data and used to predict an unknown value.

Learning Binary Decision Trees:

- Start with all data points in a single node
    - Classification: predict the most common value
    - Regression: predict the mean value
- Choose the 'best' binary decision rule, using it to split the data into two groups
    - Sort values, identify midpoints for thresholds
    - If *discrete*, use "=' and "≠" to split; if *real*, use ">" and "≤"
    - **For classification**: choose the split with the highest *information gain*
        - Information gain is the theoretic measure of how well the split separates the data

$$\text{Gain} = \frac{F((A+C),(B+D)) - F(A,B) - F(C,D)}{A+B+C+D}$$

$$\text{where: } F(X,Y) = X\log_2\frac{X+Y}{X} + Y\log_2\frac{X+Y}{Y}$$

    - **For regression**: minimize the sum of squared errors
- Repeat Step 2 on each group, until some stopping criteria is reached
- Prune the tree to remove irrelevant rules and prevent overfitting
    - Do a train/test split, train on training set, graph test set accuracy by depth of tree.
    - Ideally iterate on this with multiple splits; an obvious, or at least reasonable pattern should emerge

Advantages of Decision Trees

- Easy to learn the tree automatically from the dataset
- Generally good, though not 'top' performance on classification tasks
- Can do both classification and regression, real and discrete inputs
- Gives an idea of what variables are important in predicting the target value
    - More important variables at the top of the tree, while unimportant variables are often not included

When to Use Decision Trees

- If lots of records are available; high number of attributes is fine with plenty of records, but avoid high number of attributes and low number of records
- If the ability to *Explain* the rules of prediction is important; while not as accurate as some 'black-box' algorithms, decision trees are very transparent, which can be important, especially when applied in social contexts.
- Inductive Bias
    - Shorter trees are prefered to longer trees
    - Trees that place high information gain close to the root are prefered over those that do not
    - Flexible - any function can be represented
    - Splits on one attribute at a time
    - Splits are axis-aligned
- Suboptimal for linear functions

Other Notes on Decision Trees

- Multi-way splits (instead of binary) are okay, but binary is generally prefered. If using multi-way, use a *gain ratio* instead of information gain to choose splits, otherwise dataset gets fragmented, leading to poor generalization accuracy.
- Can be easily used to predict *multiple outputs*; create a single tree where the split criterion is the avg impurity over all outputs
- Lots of ways to handle *missing data*
    - Delete missing observations (not so good)
    - Treat "missing" as its own value (good only if missingness is informative)

- Surrogate splits (mimics the primary split, but not exact copy due to missing/incorrect information)
- Propagate examples with missing values down both branches as partial observations

3. Ensemble Methods

An *ensemble method* is a collection of models that learn multiple predictors and let them either *vote* for classification (*majority* or *soft* voting possible) or average for regression. Assuming all models are independent (ideally '*maximally independant*'), ensemble methods often achieve greater accuracy than any model alone, in much the same way as the average of a group guessing the amount of jelly beans in a jar is often closer than most particular guesses, though this is often at the expense of interpretability.

There are two natural ways to create multiple, different predictors from training data:

1. Learn different classes of models using the same training data
2. Learn the same type of model using different, randomly selected subsets of the training data

If errors are *unbiased* and *uncorrelated*, the mean squared error of the ensemble is smaller than the average MSE of the individual models by a factor of 1/M.

Reasons Why Might Ensembles Improve Accuracy

1. *Statistical* - insufficient training data to distinguish between multiple hypotheses; each might have its own biases for generalization, so best to average over these
2. *Computational* - some learners may converge on suboptimal hypotheses (e.g. stuck in a local optima); averaging helps to downweight these poor performers
3. *Representational* - the hypotheses being searched may not contain the true target function, while ensembles can create a good approximation
4. *Reduced Instability* - small changes in training data for a decision tree may lead to dramatic differences; averaging helps reduce this problem, improving both accuracy and stability

A Handful of Common Ensemble Methods

1. *Stacking*
    - Learn a bunch of different methods using the same training dataset
    - Designate a validation set in addition to train/test sets; learn an additional classifier (typically logistic regression) to choose the weights of the individual classifiers
2. *Boosting*
    - *2.1 Adaboost*
        - Most common approach
        - On each step, iteratively reweight the data using the current set of models, giving exponentially higher weight to incorrectly predicted data points, and lower weight to correctly predicted points; then, learn a new model using the reweighted data, taking a weighted avg of individual predictions
    - *2.2 Gradient Boosting*
        - Additive - on each step, fit the model to the residuals left by fitting all previous models; equivalent to gradient descent in function space
    - Advantages
        - Can start with weak classifiers and get resulting strong classifiers
        - Reduces both bias and variance
        - Minimizes convex loss function
    - Disadvantages (compared to random forests)
        - Harder to implement
        - Less robust to outliers
        - Sensitive to parameter values
        - From a computational perspective, less parallelizable
3. *Bagging*
    - Short for “bootstrap aggregation”; learn a large set of models, each using a different bootstrap sample from the training dataset, with a final prediction being an unweighted avg/vote of the individual predictors
    - E.g. sample data records (rows) uniformly at random with replacement

- Advantages
  - Higher performance than individual trees (often slightly less than boosting or random forests)
  - Increases stability, reduces overfitting
  - Trivial to parallelize
- Choose sample size (% for each aggregation) using out-of-bag ("OOB") error on a separate validation set.

4. *Random Forests*
   - Learn a large set of models (e.g. decision trees), each using a different bootstrap sample; final prediction is unweighted avg of individual predictors
   - Difference from bagging
     - Each split, restrict choice to a randomly chosen subset of features
     - Typical choice - if original dataset has *p* dimensions, restrict to *sqrt(p)*
   - Key parameter choices:
     - Number of trees in the forest
     - Whether and how to prune the trees (e.g. min number of samples per leaf, max depth, min number samples to split, etc.)
     - Number of records and features to sample
     - ^ generally fairly robust to all of thee above!

<u>Accuracy vs. Interpretability in Random Forests</u>

Big tradeoff here vs. simpler models (e.g. decision trees). Context is usually key for which to prioritize.

To interpret the model as a whole, one can post-process the ensemble in various ways to calculate the measures of *variable importance*.

- <u>Gini Importance</u>: mean decrease in node impurity across all tree splits on that variable; useful, but biased toward continuous or discrete variables with many values

$$Imp(X_m) = \frac{1}{N_T} \sum_{T} \sum_{t \in T: v(s_t) = X_m} p(t) \Delta i(s_t, t)$$

Mean over trees | Sum over splits on that var. | Proportion of data pts reaching | Decrease impurity, e. info. gair

- Permutation Importance: mean decrease in overall accuracy (as measured on OOB samples) when a given variable is permuted; biased toward correlated variables and very computationally expensive; useful for backward elimination-based procedure for variable selection
- Interesting Alternative (not formal name!)
    - Learn a single decision tree that best predicts the output of the forest; preserves interpretability along with a substantial fraction of the gains in accuracy

4. Support Vector Machines

A *support vector machine* is an optimization-based on prediction approach based used primarily for binary classification.

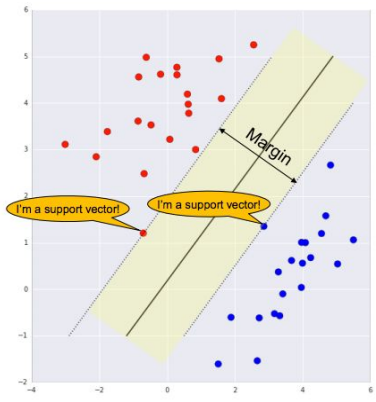
Key Idea #1: Learn a *decision boundary* that optimally separates positive and negative training examples

Key Idea #2: Learn a linear decision boundary in high dimensional space, corresponding to a non-linear decision boundary for the original problem

Assumes real-valued attributes on the same scale, thus it is very important to pre-process data before training the model. Here's how:

- Normalize real-valued attributes (either $[0,1]$ or $\mu = 0$, $\sigma^2 = 1$); use same scaling for training and test data
- Replace discrete-valued attributes with dummy-variables (usually one-hot encoding)

How to Choose a Separating Line



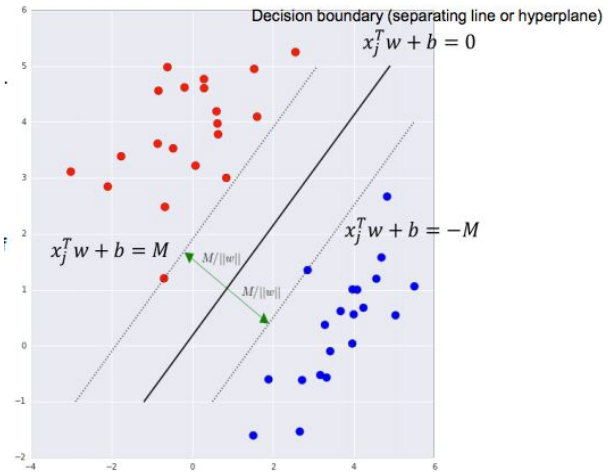
Choose the line that maximizes the *margin* between classes. Points on the margin are called support vectors.

The classifier can be defined entirely by the set of support vectors, making it 1) fast for classification of test points, 2) fast leave-one-out cross-validation 3) faster, but still expensive for training

Why is Maximizing the Margin a Good Idea?

1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
3. Backed up by statistical learning theory provable bounds on generalization error.
4. Empirically it works very well.

How to Maximize the Margin *(linear, separable cases)*



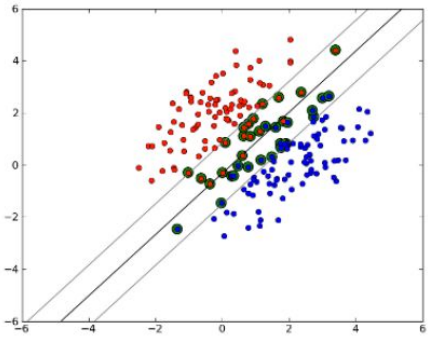
Margin = 2M / ||w|| (distance between parallel lines)

Goal: maximize margin subject to constraints for all j: $y_j (x_t^j w + b) >= M$

Simplify by change of variables, dividing w and b through by M

New Goal: minimize ||w|| subject to constraints for all j: $y_j (x_t^j w + b) >= 1$

How to Maximize the Margin *(linear, non-separable cases)*



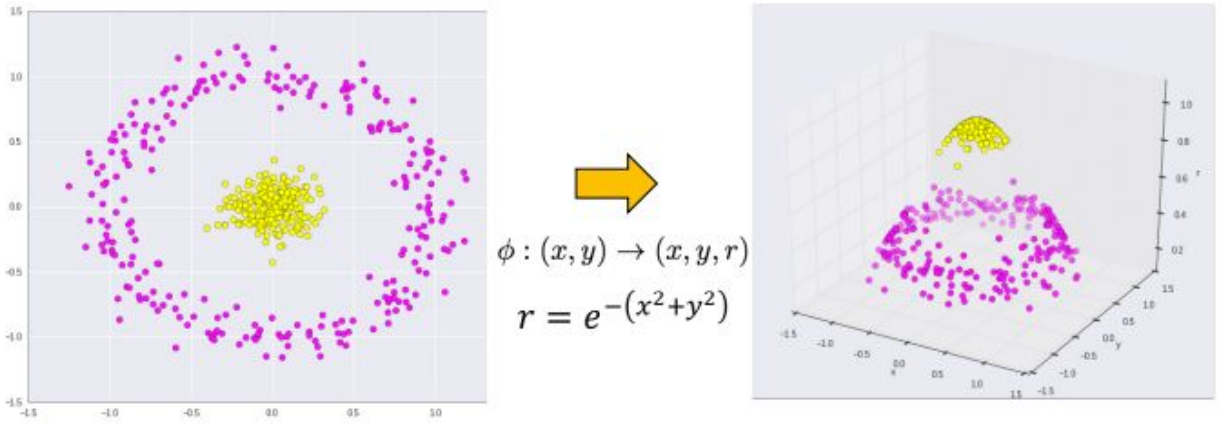
Use multiple support vectors! Add an error term into the minimization equation, such as $y_j (x_t^j w + b) >= 1 - \varepsilon_j$ where $\varepsilon_j >= 0$

Training points with $\varepsilon_j > 1$ are misclassifications.

Non-Linear Decision Boundaries

1. Map the input space to a high-dimensional feature space
2. Learn a linear decision boundary (hyperplane) in the high-dimensional space

3. Map back to lower-dimensional space, giving a non-linear boundary



'Kernel Trick' used to solve these problems in more computationally efficient (and feasible) methods - instead of mapping both x_i and x_j into a high-dimensional (possibly infinite-dimensional!) space and computing the dot-product in that space, we can compute a function $K(x_i, x_j)$ of the original data points. Here are some of common kernel functions:

Linear kernel: $\phi : x \rightarrow x \quad K(x_i, x_j) = x_i \cdot x_j$

Non-linear kernels:

- Polynomial kernel: $K(x_i, x_j) = (\gamma(x_i \cdot x_j) + r)^d$
- Sigmoid kernel: $K(x_i, x_j) = \tanh(\gamma(x_i \cdot x_j) + r)$
- Gaussian kernel: $K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$

The Gaussian kernel is often referred to as the "Radial Basis Function" (RBF). It is one of the most widely used kernels, and generally a good default option.

Variants and Extensions of SVMs

SVMs are mainly used to do non-probabilistic, binary classification. To do multi-class classification, learn a binary classifier for each class (class *k* vs. *rest*). To estimate class probabilities, logistic regression using outputs of k(k-1) pairwise SMVs (not generally recommended). SVMs can also be used for regression and anomaly detection, but not covered here.

Advantages of SVMs

- Very good performance. Generally beat just about everything but advanced CNNs
- Theoretical guarantees about generalization performance (accuracy for labeling test data) based on statistical learning theory
- Rely on convex optimization, and thus do not get stuck in suboptimal local minima (vs. neural nets, which have issues with these, and decision trees, which rely on greedy search)
- Fairly robust to the curse of dimensionality[1]
- Flexible: can choose kernel to fit very complex decision boundaries
- Generally avoids overfitting, assuming well-chosen parameters
- Fast and memory efficient, especially when number of support vectors is small

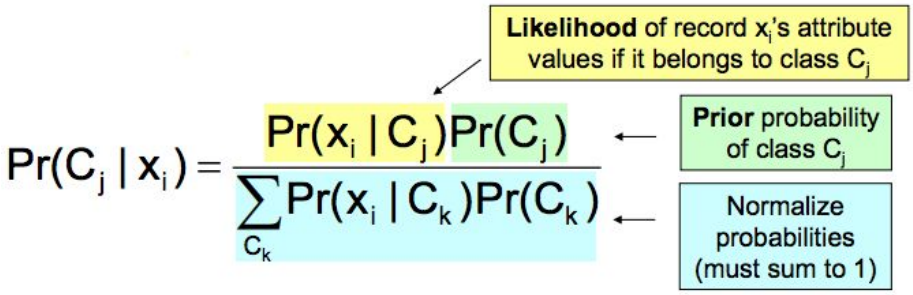
Disadvantages of SVMs

- Training is computationally expensive - dependent on number of support vectors, but generally quadratic to cubic in number of data points
- Sensitive to choice of parameters, particularly the constant C and kernel bandwidth
    - C trades off misclassification rate against simplicity of the decision surface. Low C, smooth decision surface; High C, more training examples classified correctly
    - Larger γ = lower bandwidth (increasing weight on nearest training example
    - Proper C and γ choice critical to SVM's performance; use GridSearchCV in sklearn with C and γ spaced exponentially far apart
- Not much interpretability for non-linear models; visualization can help in lower-dimension models, but becomes impossible with higher-dimensions

[1] Curse of Dimensionality - intuitive explanation from Quora post by Kevin Lacker: "Let's say you have a straight line 100 yards long and you dropped a penny somewhere on it. It wouldn't be too hard to find. You walk along the line and it takes two minutes. Now let's say you have a square 100 yards on each side and you dropped a penny somewhere on it. It would be pretty hard, like searching across two football fields stuck together. It could take days. Now a cube 100 yards across. That's like searching a 30-story building the size of a football stadium. Ugh. The difficulty of searching through the space gets a lot harder as you have more dimensions. You might not realize this intuitively when it's just stated in mathematical formulas, since they all have the same "width". That's the curse of dimensionality. It gets to have a name because it is unintuitive, useful, and yet simple."

5. Bayesian Methods

Bayesian Methods are methods that learn a probabilistic model for each class, then compute the class posterior probabilities ($Pr(C_j | x_i)$) using Bayes' Theorem.



Priors and likelihoods are obtained in two general ways - from prior knowledge (e.g. domain experts) or learned from a representative training dataset.

Naive Bayes is one very simple, but useful way to learn these models from data. It assumes that all attribute values are conditionally independent, given the class.

$$Pr(x_i | C_k) = \prod_{j=1..J} Pr(A_j = v_{ij} | C = C_k)$$

Gaussian Naive Bayes Classification - learn a Gaussian distribution from all of the training records with a given criteria, and uses this distribution to estimate the probability

Steps in Training Naive Bayes

1. Learn a class-conditional model for each attribute for each class in the training data
2. To classify a given test record, compute the likelihood of each of its attributes given each class, and multiply to obtain the total likelihood
3. Obtain the prior probability of each class from the training data, and combine this with the likelihood using Bayes Theorem

When to Use Naive Bayes

- Class prediction in a dataset (can only do classification - not regression!)
- Unlike decision trees, attributes/records ratio isn't of much concern

- Interpretable model is required, as well as an understanding of how each attribute affects our predictions
- Assumes that all attributes are conditionally independent given the class.

Bayesian Methods for Semi-Supervised Learning

If you have a small amount of labeled data and a large amount of unlabeled data, the unlabeled data can often be used to improve prediction performance, assuming *smoothness* (data points near each other tend to belong to the same class). Here are the steps typically used:

1. *Initialization Step*: based only on the labeled data, set $t = 0$ and estimate the parameters Θ_t consisting of class-conditional models $\Pr(x_j = v \mid y = k)$ for discrete-valued attributes, Gaussian mean/var $(\mu_j, \sigma_j^2) \mid y = k$ for real-valued attributes and priors $\Pr(y = k)$.
2. *Expectation Step (E)*: Assume discrete random variable $\hat{y}_i$ for each record. $\hat{y}_i = y_i$ if known; otherwise, we compute class probabilities given the current parameters of Θ_t, just like with a Naive Bayes:

$$\Pr(\hat{y}_i = k) = \frac{\Pr(y = k \mid \theta^t) \prod_j \Pr(x^j = v_{ij} \mid y = k, \ \theta^t)}{\sum_{k'} \left(\Pr(y = k' \mid \theta^t) \prod_j \Pr(x^j = v_{ij} \mid y = k', \ \theta^t) \right)}$$

3. *Maximization Step (M)*: compute new parameters Θ^{t+1} consisting of class-conditional models $\Pr(x^j = v \mid y = k)$ for discrete-valued attributes, Gaussian mean/var $(\mu_j, \sigma_j^2) \mid y = k$ for real-valued attributes, and priors $\Pr(y = k)$.
4. Iterate between E and M steps until convergence. This process is typically called EM (Expectation-Maximization).

The maximization setup is just like training a Naive Bayes, except that each data record with unknown y_i is treated as a partial observation of each class, weighted proportionally to the estimated class probabilities. The log-likelihood increases monotonically on each step.

Bayesian Methods for Unsupervised Learning

We can use the same EM algorithm to perform clustering if the data is entirely unlabeled, choosing K clusters, and partitioning N data points ($K << N$) to maximize log-likelihood.

The resulting clusters capture the natural grouping of the data (i.e. similar points are placed in the same cluster), but may not be useful for predicting particular output y_i.

The use of EM for clustering with real-valued attributes and assumed Gaussian distributions is called a Gaussian Mixture Model (GMM). But we'll get more into clustering in our next section...

6. Clustering

Clustering attempts to, given a set of N records, partition that dataset into k << N groups, such that records in the same group are similar to each other and records in different groups are dissimilar. The idea being that by characterizing certain subgroups of the data, we both better understand its structure and can more accurately predict properties of newly introduced or fragmented data.

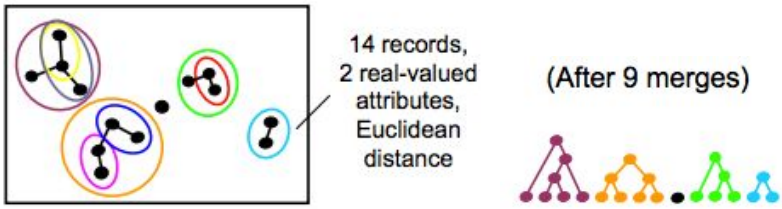
Group-Based Trajectory Modeling identifies subgroups of the population, and uses these subgroups to predict how an individual's behavior will change the course of his/her life (e.g. relevant to our studies of recidivism within the justice system).

Also commonly used in evolutionary biology to create phylogenetic trees, relating different species and showing when, in the course of their evolutionary history, their genomes diverged.

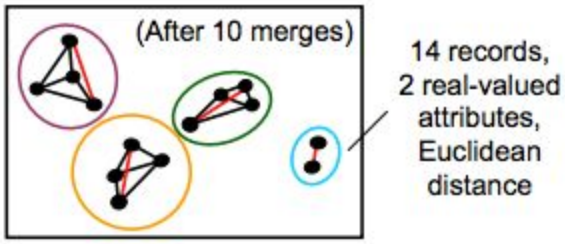
Common Types of Clustering

- *Hierarchical Clustering*
    - *Bottom-Up Hierarchical Clustering*
        - Start with N clusters, each containing one record
        - Choose the two 'nearest' clusters, and merge them into a single cluster
        - Repeat until all points are merged into a single cluster
    - *Top-Down Hierarchical Clustering*
        - Start with one clusters, containing all N records
        - Choose the *worst* cluster, and optimally partition it into two distinct clusters

        - Repeat until all points are in separate clusters
    - *Single-Link Clustering*



        - Tend to produce highly-elongated groups or “chains”
    - *Complete-Link Clustering*
        - Produces more spherical groups by considering the maximum distance between clusters



    - Advantages of Hierarchical Clustering
        - You get an entire hierarchy, not just a single cluster
        - Easy to compare different numbers of clusters
        - Bottom-Up generally prefered; some algorithms use a mix of both
- *K-means Clustering*
    - Defines an objective function that describes the quality of the groups (e.g. ‘minimize distortion’) and attempts to optimize that function
    - Assumes all attributes are real-valued, so that a centroid can be computed for any cluster (i.e. the mean value of each attribute)
    - Possible moves for *state-space search*:
        - Change position of cluster centers
        - Change mapping of point x*i* to center μ_k
        - Change number of clusters K
        - Generally, keep number of clusters fixed and iterate between the first two, then go back and play with the number of clusters (if this were embedded ‘for-loops,’ number of clusters would be the outer loop

- K-means will always converge to a solution; distortion decreases monotonically, and it will end in a state where no move will reduce the overall distortion
- K-means will *not* always find an optimal solution; for example, take a look at the following 'solved' k-means:

- How to avoid local optima in K-means?
  - Run multiple times with different start states, choosing the best result
  - Allow some moves that increase distortion, such as simulated annealing
  - Choose a start state that is less likely to result in a poor local optimum
- How to choose number of centers k?
  - Choose the k that minimizes a measure of distortion with a penalty for more complex models (thus reducing overfitting)
  - One common such criterion is the *Schwarz Criterion*, which minimizes (distortion + λk) where λ is a constant, proportional to ((# of attributes) * log(# of records)); in general, this is the best criterion with which to first test a model

<u>K-means vs. EM</u>

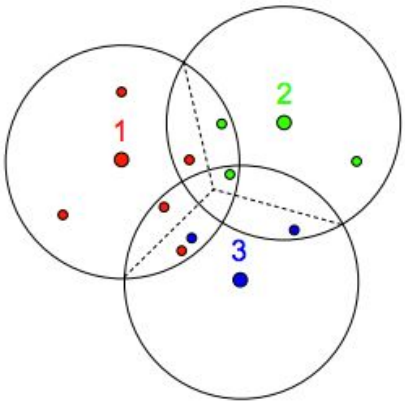
- *Assignment*: k-means makes hard cluster assignments (each point is mapped to a single cluster), while EM makes soft assignments (each point is assigned a probability distribution over clusters).
- *Modeling*: k-means models only the mean (centroid) of each cluster, while EM models the mean and covariance matrix (or with naïve Bayes assumption, the variance of each attribute).
- *Equivalence*: EM reduces to k-means when covariance matrix is diagonal, and all variances are equal and very small.

- *Speed*: k-means is much faster and consumes much less memory in practice. (the tradeoff: EM can better model the data, as measured by log-likelihood.)
- K-means is biased toward spherical clusters; EM+NB is biased to axis-aligned ellipses; EM with full covariance matrix can model non-axis-aligned ellipses.

Leader Clustering

Leader Clustering is used when a dataset is so huge that each datapoint can only be looked at once before discarding it. Here, we keep only a small set of representative points as "leaders" and summary information about the points similar to each leader.

For each data point x_i : if x_i is within distance T of any leader, add to nearest leader's group, otherwise make x_i a leader. If it falls in an overlapping region, it is randomly assigned.



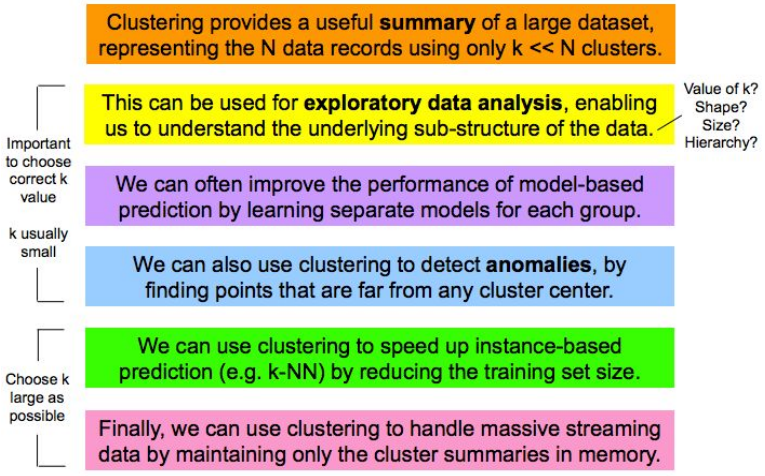
Advantages of Leader Clustering:

- Very fast - only needs to look at leaders for each point
- Guarantees group diameter < 2T, group leaders at least T apart

Disadvantages of Leader Clustering:

- Order-dependant; first point always a leader, initial clusters tend to be larger
- Points may be assigned to nearest cluster center

The Many Uses of Clustering



That's all, Folks! ✌