(14) ■ **SVMs:** Maximum margin classifier, decision boundary: $f(x) = \beta_0 + x^T\beta$. Idea is to create a margin on either side of the decision boundary to better separate the classes. ■ **Support Vectors** are the points directly on the margin and inside the boundaries but correctly classified and points that are incorrectly classified. ■ Linear SVMs as empirical risk minimization. Use Lagrange multipliers to reframe $\min_{\beta_0,\beta} \frac{1}{2}\|\beta\|_2^2 + C\|\xi\|_1$ to $\min_{\beta_0,\beta} \sum_{i=1}^n LOSS + C\|\beta\|_2^2$ called the LOSS = Hinge Loss = $[1 - y_i(x_i^T\beta + \beta_0)]_+$.

■ **Strength + Weaknesses of Linear SVMs:** (1) Does not depend on Gaussianity or any other probability distribution. (2) SVMs tend to more robust then logistic regression such as to outliers as the outlliers are only affected by a linear amount whereas in CE the loss would shoot way up and the outliers would have exponential loss. (3) $p \gg n$, SVMs are still computable and have a unique and global solution (b/c regularization). ■ **Non-Linear (Kernel) SVMs:** Recall: kernels and RKHS to take infinite dimensional non-linear problems and turn them into problems depending on n dimensions (result of Mercer's Theorem). $f(x) = \beta_0 + \sum \phi(x_i)\xi_i$.■ **Kernel Trick**: Derive optimization problem as a function of inner products $(x_i^T x_j)$ in $x's$, replace all linear inner products with $x_i^T x_j$ with $k(x_i^T x_j)$. ■ **SVM Dual Problem** (equivalent to other forms listed before): $\max_\alpha \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{i'=1}^n \alpha_i\alpha_{i'} x_i^T x_{i'} y_i y_{i'}$ subject to $\sum_{i=1}^n \alpha_i y_i = 1$, $0 \le \alpha_i \le C$, $i = 1, \dots, n$. Param $\alpha \in \mathbb{R}^{+n}$, Hyperparameter $C \ge 0$. The kernel SVM is just taking $x_i^T x_{i'}$ and replacing with $k(x_i^T, x_{i'})$, everytime we see an inner product replace it with the kernel of the inner product. ■ **Interpret**: α_i are sparse and $\alpha_i > 0$ are support vectors. Linear: $\hat\beta = \sum_{i \in SV} y_i x_i \hat\alpha_i$. Kernel: Predictions $x^* \to \hat f(x^*) = \sum_{i \in SV} y_i \hat\alpha_i\, k(x_i, x^*)$, predictions are only a function of the support vectors. ■ **Kernel:** RBF kernels are more flexible then polynomial kernels.

---

(16) ■**Unsupervised Learning:** Only have data $X_{n\times p}$ without y labels/outcomes. Goal: find some structure in the data that is likely to generalize to future data. PATTERNS via dimension reduction techniques, GROUPS via clustering techniques, ASSOCIATION via graphs/networks, ANAMOLIES, …

■ **Dimension Reduction**: Have p features and want to reduce these/ engineer new features $r \ll p$ that retain major patterns in the data. **Q.** Why use this? **A.** (1) To visualize the data o.w. w/o dimension reduction we have $\binom{p}{2}$ scatter plots? (2) Want maybe because there is lots of redundancy or irrelevant features (denoising). (3) Pattern discovery. Note that this is useful for both supervised AND unsupervised learning; However, pattern discovery leans towards unsupervised learning.

■ **Principal Component Analysis (PCA)**: Note that PCA is the optimal linear reduction technique!! $X = U\Sigma V^T$ where ΣV^T gives principal component scores (e.g. embedding of chapters) and V gives the direction, Idea is to retain high variance patterns via linear projections – linear combinations of features that maximize variance. Orthogonal projection of points onto hyperplane such that the variance is maximized. These high variance patterns retain the most information in lower dimensions! **Q**. Is the PC line the same as LS line? **A**. No, the OLS takes the vertical projection, not orthogonal projection. Set up PCA mathematically.

■ **Optimize**: Line/hyperplane = Xv , $v \in \mathbb{R}^p$ (weights on each feature) and if v is orthogonal, $v^Tv = 1$, then Xv is a linear projection and Xv becomes the **first PC** line/plane. Now we want to find the weights v that maximizes the variance of this plane. $\max_v Var(Xv)$ subject to $v^Tv = 1$. Note: $\max_v Var(Xv) = \max_v v^T Var(X) v = \max_v v^T \Sigma_{p\times p} v$ where $\Sigma_{p\times p}$ is the variance-covariance matrix; we don't actually know the population $\Sigma_{p\times p}$ so instead we will use an estimate of $\Sigma_{p\times p}$ (assuming $X \sim N(0, \Sigma)$) then $\hat\Sigma = \frac{1}{n}X^TX$ (MLE of Σ). Therefore, the optimization problem is not $\max_v v^T X^T X v$ subject to $v^Tv = 1$ (to find the first PC) and subsequent PC must be orthogonal. **All PC**: $\max_{v_k} v_k^T X^T X v_k$ subject to $v_k^T v_k = 1$ and $v_k^T v_j = 0\ \forall k \ne j$ and $k = 1, \dots, r$. These constraints are to ensure orthogonality.

■ **Solution**: Eigen decomposition. Recall, $v_1 X^T X = \lambda v_1$. PC solution is given by Eigen Decomp of X^TX . Alternatively given by Singular Value Decomp of $X = SVD(X)$.

■ **SVD**: $\max_{v_k} v_k^T X^T X v_k = \max_{u_k, v_k} u_k^T X v_k$ subject to $v_k^T v_k = 1$, $v_k^T v_j = 0$, $u_k^T u_k = 1$, $u_k^T u_j = 0$, $\forall k \ne j$. Review SVD, $X_{n\times p} = U_{n\times n} D_{n\times p} V_{p\times p}^T$, $U_{n\times n}$ are orthonormal $U^TU = I$ UNSCALED PC, $V_{p\times p}$ are orthonormal PC DIRECTIONS, $D_{n\times p}$ diagonal matrix $d_1 \ge d_2 \ge \dots \ge d_r$ s.t. $rank(X) = r$ PROP OF THE VARIANCE OF EACH PC. How to interpret SVD solution u_1 is the first major observation pattern and v_1 is the first major feature pattern. u_2, v_2 subsequent patterns are uncorrelated and note that d_1 is the first singular value and the variance explained by the first pattern is d_1^2 . ■ **PVE** (Proportion of Variance Explained): $\frac{d_k^2}{\sum_{j=1}^r d_j^2}$. ■ **CVE** (Cumulative VE - screeplot): $\frac{\sum_{j=1}^k d_j^2}{\sum_{l=1}^r d_l^2}$ by the first k PCs. ■ **Properties of PCA**: (1) The PC problem is **NON-CONVEX**, yet there exists a global unique closed form solution solved by SVD!!!!!!!!!!!!!!!!!!!!!!!!!!!! (2) u, v are unique up to a sign change. (3) The solution is ordered and nested (important for interpretation), i.e., solution is the same regardless of number of PC selected, r dimensions.

---

(17) ■**PCA (Cont.):** PCA is the best linear dimension reduction technique, $\min_{Z_{n\times k}} \|X - Z\|_F^2$ subject to $rank(Z) = k$ yields the SVD solution! PCA is very good for interpretation. ■ **Practicalities**: To center and scale or not? Application dependent! 2-models: **(Application 1)** - Cov model $X \sim N(\mu, \Sigma) \to \max Var$. This model implies we should center the features! If we scale the features, then we maximize variance patterns in the correlation matrix instead of variance-covariance matrix. If the scales are completely different such as income and proportion of race or something, then scaling could make sense! **(Application 2)** – Low-Rank Mean model. $X = LR + E$ where $E \sim iid\,(0, \sigma_\varepsilon^2)$. We don't want to center or scale here, our goal is to find the mean! **Conclusion**: If we don't assume multivariate Gaussianity then we might not want to center or scale our data. It is okay to try with and without centering and choose which has better patterns but don't scale unless the scales of features differ significantly.

■ **PCA Extensions**: Regularized PCA → functional PCA, PCA for smooth factors, spatial + timeseries. Sparse PCA → l_1 penalty. Supervised PCA (such as LDA), etc.

■ **Other Linear Dimensional Reduction Techniques**: (Although PCA is optimal linear DR, sometimes we may not only want to find the highest variance patterns). We may want to interpret different types of patterns beyond PCA. ■ **Non-Negative Matrix Factorization (NMF):** $\min_{L,R} \|X - L * R\|_F^2$ subject to $L_{ij} \ge 0$ and $R_{ij} \ge 0$ and $X_{ij} \ge 0$ so this is exactly PCA except instead of orthogonality constraints we are adding non-negativity constraints! Why do we want to do this? NMF finds patterns that go in one direction, i.e., PCA finds orthogonal directions that maximize variance, but components can mix positive and negative features, making interpretation harder. NMF finds additive, non-negative directions (parts-based patterns), making it easier to interpret components as intuitive groups or traits. These are called **Archetypal patterns**; not optimal mathematically but very interpretable.■ **Theory**: NMF yields local solution (you can get different solutions from different starting points) and is not nested/ordered – meaning choosing k is super important (change k yields diff solns). This basically yields a set of patterns that seem significant but cannot determine which is most important/significant.

■ **Non-Linear Dimension Reduction (Manifold Learning)**: Why do we want nonlinear DR? The most interesting patterns don't necessarily have to lie on a hyperplane! Project our data onto a manifold – some geometric object in r-dimensional space. Then we can visualize the data (n observations) as projected onto the manifold by unfolding this manifold into a hyperplane! ■ Manifold learning seeks a lower dimensional representation $Z_{n\times r}$, $r \ll p$, that represents the patterns/distances amongst observations.

■ **Limitations:** In Manifold learning, (1) we only get observation patterns and we cannot get/interpret feature patterns unlike Linear DR! (2) We never learn the manifold.

■ **Manifold Map**: This is the visualization of the manifold as a hyperplane. These are very good for interpretation. They are invariant to rotation or translations meaning the position does not matter because we can unfold or flip the manifold before viz as hyperplane – we only care about the relative distances between observations in manifold map. NOTE: There are two types of manifold learning (1) Global → globally represent distances or dissimilarity in lower-dimensional space; can interpret distances between two points as true distance, i.e., if two points are far we can interpret them as far! (2) Local → only locally represent distances or dissimilarities in lower-dimensional space; tries to preserve nearest neighbors so within the neighborhood, this cluster of points are close but the distance between clusters cannot be interpreted as either close or far!

■ **Kernel PCA (GLOBAL):** $D_{n\times n}$: distance or dissimilarities, $D_{ii'} = k(x_i, x_{i'})$ (rbf, poly, etc). Kernel PCA is the eigen decomposition of D (global soln)→ non-linearize distances and apply PCA.

---

(18) ■ **Spectral Methods:** Spectral methods use the eigenvectors of similarity or distance matrices to embed data in lower dimensions, capturing global structure (e.g., in PCA, spectral clustering, or Laplacian eigenmaps). **Steps**: (1) Compute $D_{n\times n}$ distance or dissimilarity matrix between observations. (2) Do PCA on $D_{n\times n} = U\Omega U^T$ and the first k columns of U are the $\hat z$. Distances → Classical MDS (Multi-Dimensional Scaling) , e.g., $D_{n\times n}$ are centered distances. How is Classical MDS similar to PCA?

■ **Theorem:** Classical MDS is exactly equivalent to PCA for Euclidean distances. Note, you can also use other distances beyond Euclidean distances such as L_1, Hamming, …

■ **Spectral methods:** PCA, Kernel PCA, Classical MDS, Spectral Embedding, Laplacian Eigenmaps, Spectral Clustering. **Not spectral methods:** t-SNE, UMAP, NMF

■ **Spectral Embedding**: D as a graph-based dissimilarity matrix (turn our observations into an $n \times n$ graph, if two observations are close draw edge, threshold = epsilon graph, nearest neighbor graphs = connect k closest neighbors). $A_{n\times n}$ is the adjacency matrix (similarity, so almost want inverse of this) and $A_{ii'}$ = edge between observations i and i'. Use Normalized Laplacian (like inverse of $A_{n\times n}$ – acts like dissimilarity for graphs) $L_{n\times n} = D_{n\times n} - A_{n\times n}$ where $D_{n\times n}$ is diagonal matrix of degrees for each node.

■ **Global (Non-Spectral) Manifold Methods:** Metric MDS, seeks to optimize a loss function that keeps distances in X (original data) and Z (dimension reduced data) close! Therefore, $\min_z \sum_{i,i'}(d_{ii'} - \|z_i - z_{i'}\|_2^2)^2$ (Euclidean distance in lower dimensional space & clearly tries to preserve global distances). CONVEX global solution but we do NOT get a nested and ordered solution.

■ **Local (Neighbor Embedding) Methods:** Find a lower dimensional space ($Z_{n\times k}$) such that close neighbors remain close, i.e., not all distance are preserved just within cluster distances are preserved. Only interpret locally! ■ **t-Stochastic Neighbor Embedding (t-SNE) , Uniform Manifold Approximation Projection (UMAP)**

■ **Steps**: (1) Calculate the original space embedding probabilities (normalized dissimilarities), e.g. for t-SNE $p_{ii'} = \frac{e^{-\frac{1}{\varphi}\|x_i - x_{i'}\|_2^2}}{norm\ term}$ and UMAP uses generalized distances and calculates distances only over nearest neighbors. (2) Define lower dimensional space embedding probabilities $q(z_i, z_{i'}) = g(|z_i - z_{i'}|)$ where t-SNE → Cauchy and UMAP → generalized Cauchy. Far away points get pushed further away!! (3) Learn Z by fitting some loss function between original and lower dimensional probabilities. t-SNE uses KL-divergence and UMAP uses Cross Entropy Loss → $\min_z \sum_{i,i'} p_{ii'} \log q(z_i, z_{i'}) + (1 - p_{ii'}) \log[1 - q(z_i, z_{i'})]$.

■ **Pro/Con:** Very useful for visualizing local patterns and finding clusters. Although they yield a local solution and are very sensitive to hyperparameter selection.

■ **Application:** Interpret UMAP with caution and JOINTLY USE THIS METHOD WITH A GLOBAL METHOD (like PCA)!!!!!

■ **Clustering**: Want to find groups of observations (or features or both) that are similar. ■ **Flat vs Nested Clusters:** A nested cluster is that smaller groups are preserved within the larger group, i.e., if we changed the hyperparameter for number of clusters it would just merge groups to create bigger clusters. Flat is just non-nested; note that flat clustering is very dependent on the hyperparameter K (the number of clusters)! ■ **Notation**: K is the number of clusters (fixed) and $C()$: $\{1, \dots, n\} \to \{1, \dots, K\}$ where $C(i)$ gives the cluster assignment for obs i and $d_{ii'}$ is the distance btw observation i and i' .■ **Goal/Optimization**: Min the within cluster dist → $\min_{C()} \sum_{k=1}^K \sum_{i:C(i)=k} \sum_{i':C(i')=k} d_{ii'}$. Group similar obs. This is NON-CONVEX (np-hard and computationally infeasible for medium sized data). Therefore, we need to find local approximations for clustering!

---

(19) ■ **Kmeans Clustering:** Gives hard clustering labels; assumes K. Goal is to approximately solve $\min_{C()} \sum_{k=1}^K \sum_{i:C(i)=k} \sum_{i':C(i')=k} d_{ii'} = \min_{C()} \sum_{k=1}^K \sum_{i:C(i)=k} \sum_{i':C(i')=k} \|x_i - x_{i'}\|_2^2$ in a fast computational manner. Use Euclidean distance (Kmeans only uses Euclidean distance). Simplify this optimization by using the mean of cluster $k \to \bar x_k = \frac{1}{n_k}\sum_{i:C(i)=k} x_i$ into $\min_{C()} \sum_{k=1}^K \sum_{i:C(i)=k} \|x_i - \bar x_k\|_2^2$ (still NP-hard). Now we apply the **Kmeans trick**! Auxiliary variable, mean of cluster $k \triangleq m_k \to \min_{C(); m_k, \forall k} \sum_{k=1}^K \sum_{i:C(i)=k} \|x_i - m_k\|_2^2$ (EQN *)

■ **Kmeans Algorithm** (iterative algo): Pseudo Code: some initialization of $\hat m_k$. While not converged: (a) minimize (EQN *) over $C()$ given $\hat m_k$ and (b) minimize (EQN *) over m_k given $C()$, end if converged. ■ **Mathematically: (b)** $\min_{m_k} \sum_{k=1}^K \sum_{i:C(i)=k} \|x_i - m_k\|_2^2 \to m_k = \bar x_k$ (sample mean of the kth cluster). **(a)** $\min_{C()} \sum_{k=1}^K \sum_{i:C(i)=k} \|x_i - m_k\|_2^2$ is harder then (b), but still simple, it is simply applying Nearest Centroid Classifier (assign points that optimize the assignment - minimize Euclidean distance - to their nearest mean m_k). Geometric solution, e.g., for 3 classes you draw a triangle with vertices at the means and edges connecting the vertices and take a decision boundary orthogonal to the midpoint of each edge. ■ **Summary**: Kmeans algorithm iterates between taking a sample mean and Nearest Centroid Classifier → very simple and fast; good for big data.

■ **Properties:** (1) Attain local solution (convergence guaranteed when means stabilize), can't certify a global solution. (2) Linear cluster boundary! (3) Very dependent on initialization, therefore we can use Kmeans++ initialization spreads out initial centroid. Worst initializations happen when all the centroids are close together and the clusters don't separate out well. (4) Kmeans will perform best when NCC is optimal or equivalently Naïve Bayes Classifier under certain conditions, i.e., Kmeans performs well under Gaussian + spherical covariance (Kmeans finds clusters that are spheres/uncorrelated features) and tends to find balanced clusters (of equal sizes). Dependent on K (not nested); changing K will often dramatically change the solution. ■ **Pro/Cons:** Advantages are quick and intuitive. Cons are the restrictive assumptions and sensitivity to K; choosing K via validation is super important! ■ **Curse**: When $p \gg n$ there exists the curse of dimensionality makes calculating distances very difficult therefore Kmeans will perform subpar if directly applied – we must first apply a dimension reduction such as PCA (to spread out the data into clusters and sphere the data by decorrelating)! Therefore, in high dimensional settings or with lots of correlated features PCA and Kmeans combination is very strong for clustering; assumption check: viz in 2d - PC1, PC2.

■ **Mixture Models**: Soft cluster labels using a generative clustering distribution (assumes underlying distribution of the data given cluster k).

■ **Gaussian Mixture Models:** This is a soft cluster extension of Kmeans! Assume the data comes from a mixture of k different distributions. Assume z_i is the cluster label for the i^{th} observation, $\pi_k = P(z_i = k) \to$ mixture probability ($\sum \pi_k = 1$). GMM assumes that $x_i | z_i = k \sim N(\mu_k, \sigma_k^2)$ (assume here that we have a spherical covariance). Estimation via MLE!! $p(x) = \sum_{k=1}^K \pi_k p_k(x)$ where $p(x) \propto \exp\left\{-\frac{1}{2\sigma_k^2}(x - \mu_k)^T(x - \mu_k)\right\}$. Hence, $l(\pi_k, \mu_k, \sigma_k^2) = \sum_{i=1}^n \log[\sum_{k=1}^K \pi_k p_k(x_i)]$ → **NOT-CONVEX** – multiplied by two parameters that we are trying to optimize. ■Trick – it is easy to compute the MLE if z_i is known, just standard computation. So let's assume that z_i is a latent or hidden variable → auxiliary variable → ■ **Expectation-Maximization (EM) Algorithm** (Iterative algo): (1) E-step → infers the hidden parameters given the parameters, $z_i | \hat\pi_k, \hat\mu_k, \hat\sigma_k^2$ (2) M-step → estimate the parameters given the inferred hidden variables $\pi_k, \mu_k, \sigma_k^2 | z_i$. ■ **E-step**: $\delta_{ik} = P(z_i = k | x_i, [\hat\pi_k, \hat\mu_k, \hat\sigma_k^2]_{t-1}) \to \hat\delta_{ik} = \frac{\hat\pi_k * P\left(x_i | [\hat\pi_k, \hat\mu_k, \hat\sigma_k^2]_{t-1}\right)}{\sum_{k'=1}^K \hat\pi_{k'} * P\left(x_i | [\hat\pi_{k'}, \hat\mu_{k'}, \hat\sigma_{k'}^2]_{t-1}\right)}$, hence we get $\hat\delta_{ik}$ which is called the responsibilities (same for all mixture models not just GMM) that cluster k takes for observation i which is just the soft cluster assignment for observation i. ■ **M-step:** Is the weighted MLE of the Gaussians: $\hat\pi_k = \frac{1}{n}\sum_{i=1}^n \hat\delta_{ik}$, $\hat\mu_k = \frac{\sum_{i=1}^n \hat\delta_{ik} x_i}{\sum_{i=1}^n \hat\delta_{ik}}$ (weighted sample mean), $\hat\sigma_k^2 = \frac{1}{n-k} \frac{\sum_{i=1}^n \hat\delta_{ik}(x_i - \hat\mu_k)}{\sum \hat\delta_{ik}}$ (weighted sample variance).

■ **Kmeans and GMM are Linear Clustering Methods!** These methods require the usage of Euclidean Distance!!!!!! Cannot use other distance metrics.

(20) ■ **Properties of EM Algorithm:** (1) Monotonically increases the observed data log-likelihood $l(\pi_k, \mu_k, \sigma_k^2)$ meaning it always converges! (2) Gives local solution, i.e., it depends on the initialization. For GMM, we initialize to the Kmeans solution. (3) Converges slowly; requires many iterations. (4) Depends heavily on K.

■ **Q:** What if the clusters have non-linear decision boundaries? **A:** Perform Kmeans after non-linear dimension reduction!

■ **Spectral Clustering:** This is just applying Kmeans after applying Spectral Embedding as a dimension reduction. Kmeans to the k smallest eigenvectors of L that don't correspond to all $1's$ eigenvector → If a graph has k connected components, then the k smallest eigenvectors of L (other than 1s) give the indicators of the components.

■ **Properties:** (1) Very nonlinear clusters. (2) Flexible. (3) Works for different shaped clusters.

■ **Hierarchical Clustering:** NESTED CLUSTERING. Family of clusters $K = 1, \dots, K = n$. Visualize nested clusters using a **dendrogram** – binary rooted tree. Shows family from $K = 1$ at the top of the tree to $K = n$ at the bottom of the tree; terminal leaf – has a single sample in each cluster. Building a dendrogram GREEDY ALGORITHM: Top down (divisive clustering) or bottom up (agglomerative clustering – start at the bottom with each single observation and iteratively merge/group two closest observations and repeat until at the root - faster for computation!). The order of merging on dendrogram is represented by the height of the merge in the tree! There are $n - 1$ steps/levels on dendrogram. ■ **Properties:** (1) since GREEDY (approx.) → local solution. (2) Height matters → distance or dissimilarity at which objects are merged. (3) X-axis has ordering and matters but can be flipped/reflected. (4) The number of clusters are given by a horizontal "cut" across the dendrogram – good interpretation of cluster assignments!

■ **Details:** Input, $D_{n\times n}$ distance/dissimilarity matrix. ***Q***. How to merge objects? **A**. Use LINKAGES → dissimilarity metric used to merge objects of one or more observations.

■ **Linkage**: How we merge sets of observations! ■ **Single Linkage** (Min), $d(A,B) = \min_{i \in A, i' \in B} d_{ii'}$ ■ **Complete Linkage** (Max), $d(A,B) = \max_{i \in A, i' \in B} d_{ii'}$ ■ **Average Linkage**, (pairwise average of single and complete linkages) $d(A,B) = \frac{1}{n_A n_B}\sum_{i\in A}\sum_{i' \in B} d_{ii'}$ → weighted average pairwise dissimilarity. ■ **Centroid Linkage,** $d(A,B) = dist(\hat\mu_A, \hat\mu_B)$

■ **Ward (∈Centroid),** Based upon Euclidean distances – weighted centroid linkage. $d(A,B) = \frac{n_A n_B}{n_A + n_B}\|\hat\mu_A - \hat\mu_B\|_2^2$.

---

(21) ■ **Properties of Linkages:** ■ **Single Linkage:** (1) Gives chaining, i.e., just add on single observation to a big cluster iteratively. (2) Chaining is flexible and good at handling different cluster shapes if the points are close together in their clustered shapes. (3) Good for outlier detection! If an observation has high height in the dendrogram, it means it was joined very late to the cluster, so it is very dissimilar or far from other observations. ■ **Complete Linkage:** (1) Optimal for spherical clusters/ balls of uncorrelated features and balanced clusters. (2) Similar in properties to Kmeans and GMM however it is most robust to outliers! Whereas Kmeans is sensitive to outliers.

■ **Average Linkage:** In between single and complete. (1) Good for data that is close to spherical and balanced clusters. (2) Has statistical consistency – on average this gives the best clustering! ■ **Ward Linkage:** (1) Gives balanced and well separated clusters. (2) ISSUE – can lead to inversion – flat parts of the dendrogram – bad property but tend to happen lower down in the tree so maybe it doesn't matter as much. ■ **NOTES:** When we have very correlated features all methods perform kind of bad – apply PCA first.

■ **NOTE:** THAT IN ORDER TO GET THE CLUSTER FOR SOME K WE "CUT" THE DENDROGRAM!!! DENDROGRAM = GOOD VIZ DATA SUMMARY.

■ **Bi-Clustering:** Cluster both the observations and the features. Can use a cluster heatmap – hierarchical clustering on features and observations separately and reorder the rows and columns according to the dendrogram order and plot the dendrograms on the x and y axis of the heatmap and fill in the heatmap according to some color gradient. We will see some good block like patterns if the data has groups of observations and features. Strong interpretations for both observations and features!

■ **Validation:** Clustering → choosing K the number of clusters in the data. Goal: Minimize the within cluster distance/dissimilarity. Plot the within cluster distance across all values $K = 1, \dots, n$. For $K = 1$ the within cluster distance is high and decreases to $K = n$ where each observation is its own cluster and the within cluster distance is 0. This is like a training error curve and ends up not being useful for choosing the value of K. No simple way to get test error. So, this is a problem, and we need to try something else!

■ **Silhouette Statistic:** A measure of the goodness of cluster for each observation i – HIGHER IS BETTER = most observations well clustered. Let a_i = average within cluster dist for obs i and b_i = average between cluster dist for obs i. The silhouette stat is $S_i = \frac{b_i - a_i}{\max\{a_i, b_i\}} \in [-1,1]$. If $S_i \to 1$, then the average between cluster dist is large and the average within cluster dist is small hence the observation is well clustered. If $S_i \to -1$, then the observation is poorly clustered. Can choose $\hat K = \operatorname{argmax}_K \frac{1}{n}\sum_{i=1}^n S_i$.

■ **ISSUES of Silhouette Stat:** (1) This yields strange behavior for $K = 1, 2$ and $K = n$, so useful if we try to determine if there are 3 to 10 clusters but can break if we are trying to decide 1 or 2 clusters. (2) Performs well for spherical and balanced clusters (okay for Kmeans, GMM, Complete, Wards). (3) NEVER use this for single linkage, spectral clustering, or nonlinear dimension reduction technique + a method like Kmeans or GMM. - Silhouette score assumes compact, convex clusters and uses Euclidean distances in the original space—this causes misleadingly low scores for methods like single linkage (which forms chained clusters), spectral clustering (which relies on graph-based similarity), or clustering after nonlinear dimensionality reduction like UMAP (which distorts distances and loses true structure). (4) Can help identify outliers too!

■ **Stability** (Consensus clustering): Idea: retain stable + reliable interpretations. Approach: repeated subsample of data and apply clustering with fixed K. Record the co-cluster membership → Consensus Cluster Matrix $C_{n\times n}$ where entry $C_{ii'} = \frac{\#\ times\ i+i'\ clustered\ together}{\#\ times\ i+i'\ sampled\ together}$. If viz, want strong diagonal of 1s and off diagonal 0s = stable clustering across samples. Can compute variance of cluster assignment for observation $i \to U_i = \sum_{i'=1}^n C_{ii'}(1 - C_{ii'})$. Then we can take $\hat K = \operatorname{argmin}_K \frac{1}{n}\sum_{i=1}^n U_i$ – choose K with smallest variance.

■ **Note:** We may have some clusters that are very stable on the diagonal as the within cluster variance is very low but the other clusters for said K are less stable. This may indicate that if K is smaller, there is a cluster within a cluster (family/embedded cluster)!!!! Note that this stability method is done using Bootstrap Sampling to subsample!

■ **Adaptive Basis Function Models** (Supervised Learning): Regression $y = f(x) + \varepsilon$. Now, $f(x) = \beta_0 + \sum_{m=1}^M \beta_m \phi_m(x, \varphi_m)$ where ϕ_m is a basis function learned from our data (φ_m are any parameters of the basis function). Here we are trying to learn weights β_m and basis function ϕ_m from the data. Such classes of supervised models are: (1) Decision Trees where ϕ_m are indicator functions (non-linear regression function). (2) Neural Networks where ϕ_m are other learned models (learn a model within a model).

---

(22) ■ **Decision Trees:** $\phi_m = \mathbb{I}\{x \in R_m\}$ where R_m is some rectangular region of our domain. → fit + viz model as a binary rooted tree. → Similar approach to hierarchical clustering – local greedy approximation (optimal decision at each step without regard to previous step) →start at top of tree and split data based on y. The optimal tree is NP-hard; therefore, we want a fast approximation using a GREEDY APPROACH = Recursive Binary Partitioning (Divisive/Top-Down). Top-Down is faster computationally.

■ **Theory:** For each split: $\min_{j,t}\left[\min_\varphi \sum_{i\in R(t,j)} L(y_i; \varphi * \mathbb{I}\{x_i \in R(t,j)\}) + \sum_{i \notin R(t,j)} L(y_i; \varphi * \mathbb{I}\{x_i \notin R(t,j)\})\right]$. Want to minimize some type of loss where the observation is in this region vs it is not in this region. ■ **Example:** Loss → Regression: MSE $L(y, \varphi) = \|y - \varphi\|_2^2$, prediction for region $\hat\varphi_m = \frac{1}{|R_m|}\sum_{i \in R(t,j)} y_i$ → sample mean for m^{th} region.

■ **Brute Force:** Try all p features and all possible split points. We can make this fast via a few CS tricks to reduce computation time – Faster than other ML algorithms!!

■ **Classification** ($y_i \in \{1, \dots, K\}$): The loss $L(y, \pi_{1:K}) \to$ multi-class $\to \hat\pi_k = \frac{1}{|R_m|}\sum_{i \in R_m} \mathbb{I}\{y_i = k)\}$ which is simply the proportion of observations in each class. Can use

■ **Misclassification Loss:** $\frac{1}{|R_m|}\sum_{i\in R_m} \mathbb{I}\{y_i \ne \hat y_i)\}$, is fine to use but maybe not the best here as it does not lead to pure nodes! This is clearly not ideal.

■ **Cross Entropy Loss:** $\sum_{k=1}^K \hat\pi_k \log \hat\pi_k$, (binary case = $\hat\pi_k \log \hat\pi_k - (1 - \hat\pi_k)\log(1 - \hat\pi_k)$ form for logistic regression). The issue is that it is slower computationally!

■ **Gini Loss** (in between misclass and CE): $\sum_{k=1}^K \hat\pi_k(1 - \hat\pi_k)$, is far faster to update and leads to more pure nodes then misclassification loss!

■ **Size:** The largest possible tree is $\le n \to$ pure node tree (training error is 0 = overfit). Can grow large tree and then "prune" (cost-complexity prune = regularization tree size).

■ **Properties:** (1) Trees can handle mixed types of features (continuous + binary/categorical + ordered). Don't need tons of preprocessing like one-hot encoding. (2) Missing values, can use as categorical feature, can use surrogate splits (mimics split), etc. (3) Interpretation of features using ■ **MDI (Mean Decrease in Impurity)** - difference in loss before the split and after the split for all splits. (4) Feature can be split on multiple times in a Decision Tree. (5) Bad for linear decision boundaries – struggle with lines!

■ **Results**: Decision Trees are not good predictors because they have high variance! The bias of tree depends on the size of the tree – if the tree is very deep, the bias will decrease and if the tree is deep, it will have high bias. If the decision boundary is linear the tree will incur a lot of bias. If the tree is large, the variance is very large! For moderate trees the variance is still high → variance is always high. Weak learners = poor predictors but better than random chance. Use many trees and LLN to reduce variance.

■ **Bagging** (Bootstrap aggregation): x^{*bag} → bootstrap sample (w/ replacement). $f^{*bag}(x) = \frac{1}{B}\sum_{b=1}^B \hat f_b(x^{*bag})$ → averaging over many $\hat f$ learners fit to many bootstrap samples. The bias stays the same and the variance is reduced! This is an ensemble method.

---

(23) ■ **Out-of-Bag Error** (OOB Error): Error of all ensemble members that didn't contain observation i evaluated at observation i, e.g., in a random forest is the error estimate computed by evaluating each training sample on the subset of decision trees for which that sample was *NOT* included in the bootstrap sample. Like LOOCV error. Built in validation method! To tune a random forest using OOB error, train multiple models with different hyperparameters and choose the one that performs best on data points it didn't see during training — as measured by its out-of-bag error.

■ **Theorem:** Bagging offers no reduction in variance for linear regression + classification models when the true underlying model is linear. Gauss-Markov Theorem.

■ **Properties:** When does bagging work? Works under non-linear models and methods. Typically, only used when fitting is very fast (or else computation is very expensive).

■ **Law-of-Large-Numbers (LLN):** Assumes iid. Are bagged trees are correlated. Therefore, to reduce variance we need to decorrelate. This is the idea behind RF.

■ **Random Forest:** Idea: Build an ensemble of decorrelated "random trees" (differ from Decision Trees because we only consider a random subset of features in the split).

■ **Algorithm**: for $b = 1, \dots, B \to$ bootstrap sample x^{*bag} . Build a random tree and at each split, consider a random subsample of features to split upon. Use the same tree building algorithm. Then take the ensemble average of all random forest trees. NOTE – RF is like an **Adaptive KNN Algorithm**. Average many decision boundaries = smooth.

■ **Notes:** Each random tree is a weak learner (non-optimal) but the idea, like ridge regularization, is that the BIAS increases (because we split on random subset of features), but the variance is reduced dramatically b/c we have a diverse ensemble of trees. NOTE - points should usually in terminal leaf with points that they are close to!

■ **Claim:** The random forest is the BEST off the shelf predictor for tabular data – don't have to do too much tuning, it just tends to work! Very hard to overfit.

■ **Hyperparameters:** (1) Tree depth: Large trees are favorable to drive down the bias (default = p). (2) B = # of random trees. Use the OOB error (when it flattens out) to decide what value B. The smaller the tree depth, the more trees we need to fit! (3) m = max_features = # of features in subsample to split upon, if m is very large ($m \to p$) then this is just the same as bagging and if m is very small the performance will significantly degrade! The default m = max_features = $\sqrt{p}$.

■ **Feature Importance:** Take the average MDI over all RF Trees. RF is great for feature importance! ■ **Prediction:** $\hat f^{RF}(x) = \frac{1}{B}\sum_{b=1}^B \hat f^{*RF}(x^{*bag})$ for new datapoint we feed it through each of the RF tree predictors and take the average. ■ **Random Forest Advantages:** Great predictors (hard to overfit), strong interpretations but lose the tree-based interpretations of predictions, works for all data types and responses, great for missing data – Miss Forests → used for missing data imputation.

■ **Random Forest Disadvantages:** (1) $p \gg n$ and we have a lot of irrelevant features, need m = max_features to be large. (2) Highly correlated features have okay prediction, but it ruins MDI interpretation of the features!

■ **Intro to Model Stacking:** Idea: Diverse ensembles are great predictors! Ensemble members can be different types. Try to find the optimal $\hat f$ but why not take the ensemble of many $\hat f$ and take a weighted average (because some maybe better than others and should be weighed higher): $\min_w L\left(y; \frac{1}{M}\sum_{m=1}^M w_m \hat f_m(x)\right)$. We need to learn the weights $w_1, \dots, w_M$ on new data to avoid overfitting – use training and validation to learn $\hat f$ and a QUERY set to learn the model stacking weights! (In RF $w_1 = w_m = 1$).

---

(24) ■ **Boosting:** Sequentially/adaptively learn ensembles → fitting to residuals; stage wise or greedy modeling. Intuition is to slowly fit residuals to hopefully outperform.

■ **AdaBoost:** $y_i \in \{-1,1\}$; $i = 1, \dots, n$. Algorithm: start with $\hat f_0 = 0$ and weights $w_i = \frac{1}{n}$. Repeat for $m = 1, \dots, M$: **(a)** fit a weighted classifier $(w_i y_i; f_m(x_i))$. **(b)** weighted misclassification error: $\widehat{err}_m = \sum_{i=1}^n w_i I\left(y_i \ne \hat f_m(x_i)\right) / \sum_{i=1}^n w_i$. **(c)** Update ensemble coefficients $\hat\alpha_m = \log\left(\frac{1 - \widehat{err}_m}{\widehat{err}_m}\right)$. **(d)** Update weights: $w_i = w_i \exp\left\{\hat\alpha_m I\left(y_i \ne \hat f_m(x_i)\right)\right\}$ and renormalize s.t. $\sum w_i = 1$. End loop. Output = $sign(\sum_{m=1}^M \hat\alpha_m \hat f_m(x))$. ■ **Intuition:** (c) yields log odds of weighted misclassification error. If $\hat f_m$ is a good classifier the misclassification error is low and the $\hat\alpha_m$ is high (give more weight to the ensemble). (d) Suppose the i^{th} observation is misclassified and $\hat\alpha_m$ is large, then w_i increases significantly; if it is classifier correctly then it's just $e^0 = 1$ so we keep the previous weight, but it normalizes so it down weighs all correctly classified observations! FITTING TO MISCLASSIFIED POINTS AT EACH STAGE which is equivalent to fitting the residuals. Use Decision Trees as a base learner and shallow trees because they are very fast and weak learner → weak learners make mistakes which is great for sequential learning. Avoid overfitting with small learning rate and weak learners!

■ **Forward Stage-Wise Additive Modeling** (FSAM) (Greedy, Sequential learning)**:** Repeat for $m = 1, \dots, M$, (a) $\hat\beta_m, \hat h_m() = \operatorname{argmin}_{\beta, h()} L(y; \hat f_{m-1}(x) + \beta h(x))$. (b) $\hat f_m(x) = \hat f_{m-1}(x) + \hat\beta_m \hat h_m(x)$. End loop. ■ **Intuition:** At each stage, fit base learner to the residual from the previous stage $L\left(y; \hat f_{m-1}(x)\right)$; Not optimal because its fast to learn weight and learner! So can just fix weight as ε = hyperparameter to slow it down; $\varepsilon \to 0$ slow!! ■ **Claim:** AdaBoost is FSAM with an exponential loss function (upper bound of CE).

■ **Epsilon Boosting** (Slow Boosting)**:** FSAM where learning rate or weight is fixed to a small constant ε (hyperparameter): $\hat h_m() = \operatorname{argmin}_{h()} L(y; \hat f_{m-1}(x) + \varepsilon * h(x))$.

■ **Gradient Boosting:** Idea: epsilon boosting but go slower! Do not completely fit a full learner $\hat h_m()$ at each stage but instead go downhill in the negative gradient direction. $r_m = -\left.\frac{\partial L(y, f(x))}{\partial f(x)}\right|_{f(x) = \hat f_{m-1}(x)}$ → negative gradient eval at current fit. $\hat g_m() = \operatorname{argmin}_{g()} L(r_m; g(x))$ replace response with the residual at each stage! Note that r_m is equivalent to the residual, e.g. $L() = MSE \to \frac{1}{2}\|y - f\|_2^2 \to -\frac{\partial L}{\partial f} = y - f$ (residual!). e.g., $L() = CE$ (Logistic), $\pi(x) = \frac{1}{1 + \exp\{-f(x)\}} \to L(y, f(x)) = y\log[\pi(x)] + (1 - y)\log[1 - \pi(x)] \to -\frac{\partial L}{\partial f} = y - \frac{1}{1 + \exp\{-f(x)\}} = y - \pi(x)$. ■ **Hyperparameters for GBM:** Requires careful tuning. (1) Learning rate: $\varepsilon > 0 \to$ generalize poorly and $\varepsilon < 0 \to$ very slow. (2) number of iterations = M, if we keep fitting residuals, we will eventually get training error = 0. (3) Base learner: want weak learner (like decision tree with shallow depth); if the base learner is a saturated tree, the training error = 0 and GBM stops learning after $M = 1$ (one iteration).

■ **NOTE:** GBM w/ tree stump = Generalized Additive Model. Therefore, if max_depth=2 outperforms max_depth=1 = tree stump we can claim that there are interaction terms