Data mining - Summary

Data exploration

PCA

Principal Component Analysis (PCA)

- Typically applied to reduce the number of dimensions of data (feature selection)
- The goal of PCA is to find a projection that captures the largest amount of variation in data
- Given N data vectors from n-dimensions, find k<n orthogonal vectors (the principal components) that can be used to represent data
- Works for numeric data only and it is affected by scale, so data usually need to be rescaled before applying PCA

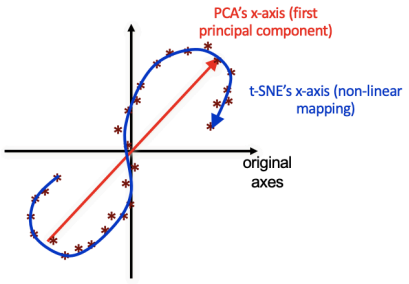
Principal Component Analysis (PCA)

- Steps to apply PCA
  - Normalize input data
  - Compute k orthonormal (unit) vectors, i.e., principal components
  - Each input data point can be written as a linear combination of the k principal component vectors
- The principal components are sorted in order of decreasing "significance" or strength
- Data size can be reduced by eliminating the weak components, i.e., those with low variance.
- Using the strongest principal components, it is possible to reconstruct a good approximation of the original data)

t-SNE

Non-linear Dimensionality Reduction

- Data in high dimensions never fills the entire space and always lives within some lower-dimensional manifold
- t-SNE is a non-linear dimensionality reduction technique used to map high-dimensional data into 2 or 3 dimensions



- Points from original space mapped onto "map points" in 2D/3D
- Unlike PCA, the mapped points are not linear combination of original attribute values, and the axes of mapped space are not linear combination (rotation) of original axes

t Distributed Stochastic Neighbor Embedding

- t-SNE tries hard to preserve local distances to nearby points
- Unlike PCA which tries to preserve global (long range) distances between points as much as possible
- t-SNE converts distances between data points to joint probabilities then models original points by mapping them to low dimensional map points such that position of map points conserves the structure of the data
- i.e. similar data points are modelled by nearby map points while dissimilar data points are modelled by distant map points

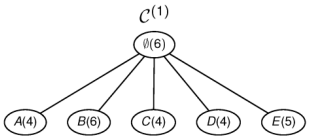
Association Rules

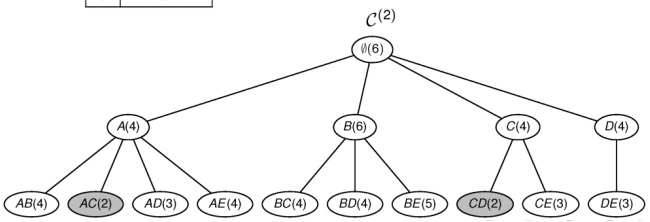
Algorithms

Apriori Algorithm

Apriori Algorithm Example (min sup=3)

D	
t	$\mathbf{i}(t)$
1	$ABDE$
2	BCE
3	$ABDE$
4	$ABCE$
5	$ABCDE$
6	BCD





First two levels of the search for minsup = 3

Apriori's Performance Bottlenecks

- The core of the Apriori algorithm
  - Use frequent (k–1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- It may need to generate huge candidate sets
  - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
  - To discover a frequent pattern of size 100 needs to generate $2^{100} \sim 10^{30}$ candidates.
  - Multiple scans of database, it needs (n+1) scans, n is the length of the longest pattern

The Eclat algorithm

The Eclat Algorithm

- Leverages the tidsets directly for support computation. The support of a candidate itemset can be computed by intersecting the tidsets of suitably chosen subsets.
- Given t(X) and t(Y) for any two frequent itemsets X and Y, then t(XY)=t(X) ∧ t(Y)
- And sup(XY) = |t(XY)|

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

(a) Binary database

t	i(t)
1	*ABDE*
2	*BCE*
3	*ABDE*
4	*ABCE*
5	*ABCDE*
6	*BCD*

(b) Transaction database

x	A	B	C	D	E
t(x)	1	1	2	1	1
	3	2	4	3	2
	4	3	5	5	3
	5	4	6	6	4
		5			5
		6			

(c) Vertical database

The FP-Growth Algorithm

The FP-Growth Algorithm

- Leave the generate-and-test paradigm of Apriori
- Data sets are encoded using a compact structure, the FP-tree
- Frequent itemsets are extracted directly from the FP-tree

- Major Steps to mine FP-tree
  - Construct the frequent pattern tree
  - For each frequent item i compute the projected FP-tree
  - Recursively mine conditional FP-trees and grow frequent patterns obtained so far
  - If the conditional FP-tree contains a single path, simply enumerate all the patterns

Benefits of the FP-Tree Structure

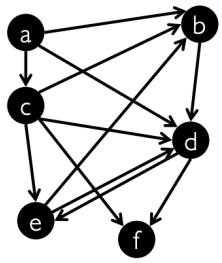
- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)

Why is FP-Growth the Winner?

- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- Other factors
  - No candidate generation, no candidate test
  - Compressed database: FP-tree structure
  - No repeated scan of entire database
  - Basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

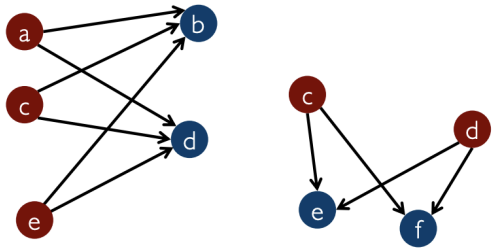
Mining Bipartite

Example



- Itemsets:
  - a = {b,c,d}
  - b = {d}
  - c = {b,d,e,f}
  - d = {e,f}
  - e = {b,d}
  - f = {}

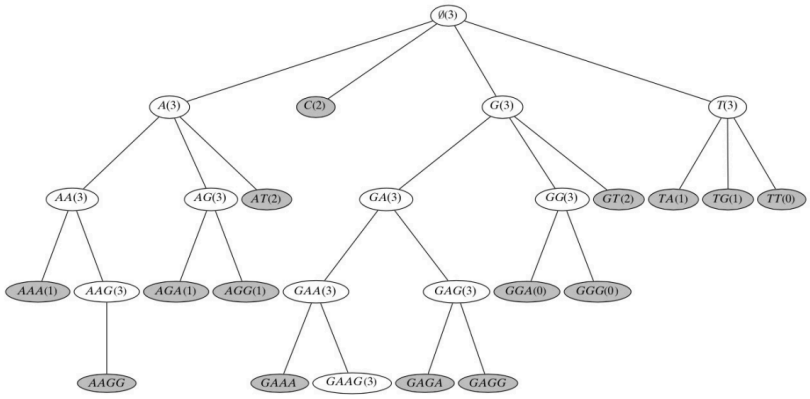
- Support threshold s=2
  - {b,d}: support 3
  - {e,f}: support 2
- And we just found 2 bipartite subgraphs:



GSP Algorithm

Level-Wise Sequence Mining: GSP Algorithm

- Searches the sequence prefix tree using a level-wise (breath-first search)
- Given the set of frequent sequences at level k, the algorithm generates the candidate for level k+1 and compute the support of each candidate and prune the not frequent ones
- For each sequence s_i in D, we check if a candidate r is a subsequence of s, if it is the support of r is incremented. Once the frequent sequences at level k are computed the k+1 candidates are generated
- For each leaf r_a, the sequence is extended with the last symbol of any other leaf r_b that shares the same prefix (it has the same parent) so $r_{ab} = r_a + r_b[k]$. If r_{ab} is infrequent, we prune it.



The search space for frequent sequences. Figure 10.1 of the textbook.

Frequent Sequences

Frequent Sequences

- Given a database D $\{s_1, \ldots, s_N\}$ of N sequences and a sequence r, we define the support count of r as

  $sup(r) = |\{s_i \mid r \text{ is a subsequence of } s_i\}|$

- And the support (or relative support),

  $rsup(r) = sup(r)/N$

Example

- Let I = {1, 2, 3, 4, 5, 6, 7, 8, 9}.
- The sequence ⟨{3}{4, 5}{8}⟩ is contained in (or is a subsequence of) ⟨{6} {3, 7}{9}{4, 5, 8}{3, 8}⟩
- In fact, {3} ⊆ {3, 7}, {4, 5} ⊆ {4, 5, 8}, and {8} ⊆ {3, 8}
- However, ⟨{3}{8}⟩ is not contained in ⟨{3, 8}⟩ or vice versa
- The size of the sequence ⟨{3}{4, 5}{8}⟩ is 3, and the length of the sequence is 4.

Example

- Given the following sequence database and a minsup of 2

SID	Sequence
10	<(a)(abc)(ac)(d)(cf)>
20	<(ad)(c)(bc)(ae)>
30	<(ef)(ab)(df)(c)(b)>
40	<(e)(g)(af)(c)(b)(c)>

- <(a)(bc)(d)(c)> is a subsequence of <(a)(abc)(ac)(d)(cf)>
- <(ab)(c)> is a frequent sequence or a sequential pattern

CBA algorithm

Mining Association Rules for Classification

- Association rule mining assumes that the data consist of a set of transactions. Thus, the typical tabular representation of data used in classification must be mapped into such a format.
- Association rule mining is then applied to the new dataset and the search is focused on association rules in which the tail identifies a class label

  $X \Rightarrow c$ (where c is a class label)

- The association rules are pruned using the pessimistic error-based methods
- Finally, rules are sorted to build the final classifier

Metrics

$$\{\text{Bread}\} \Rightarrow \{\text{Milk}\}$$

Support

Fraction of transactions that contain both X and Y

$$s = \frac{\sigma(\{\text{Bread}, \text{Milk}\})}{\#\text{ of transactions}} = 0.38$$

Confidence

Measures how often items in Y appear in transactions that contain X

$$c = \frac{\sigma(\{\text{Bread}, \text{Milk}\})}{\sigma(\{\text{Bread}\})} = 0.75$$

Lift

- Lift is the ratio of the observed joint probability of X and Y to the expected joint probability if they were statistically independent,

  $lift(X \Rightarrow Y) = sup(XUY)/sup(X)sup(Y) = conf(X \Rightarrow Y)/sup(Y)$

- Lift is a measure of the deviation from stochastic independence
  (if it is 1 then X and Y are independent)

- Lift also measure the surprise of the rule. A lift close to 1 means that the support of a rule is expected considering the supports of its components.

- We typically look for values of lift that are much larger (i.e., above expectation) or much smaller (i.e., below expectation) than one.

Maximal Frequent Itemsets

- A frequent itemset X is called maximal if it has no frequent supersets

- The set of all maximal frequent itemsets, given as $M = \{X \mid X \in F \text{ and } !\exists Y \supset X, \text{ such that } Y \in F\}$

- M is a condensed representation of the set of all frequent itemset F, because we can determine whether any itemset is frequent or not using M

- If there is a maximal itemset Z such that $X \subseteq Z$, then X must be frequent, otherwise Z cannot be frequent

- However, M alone cannot be used to determine sup(X), we can only use to have a lower-bound, that is, $sup(X) >= sup(Z)$ if $X \subseteq Z \in M$.

Closed Frequent Itemsets

- An itemset X is closed if all supersets of X have strictly less support, that is,

  $\sup(X) > \sup(Y)$, for all $Y \supset X$

- The set of all closed frequent itemsets C is a condensed representation, as we can determine whether an itemset X is frequent, as well as the exact support of X using C alone

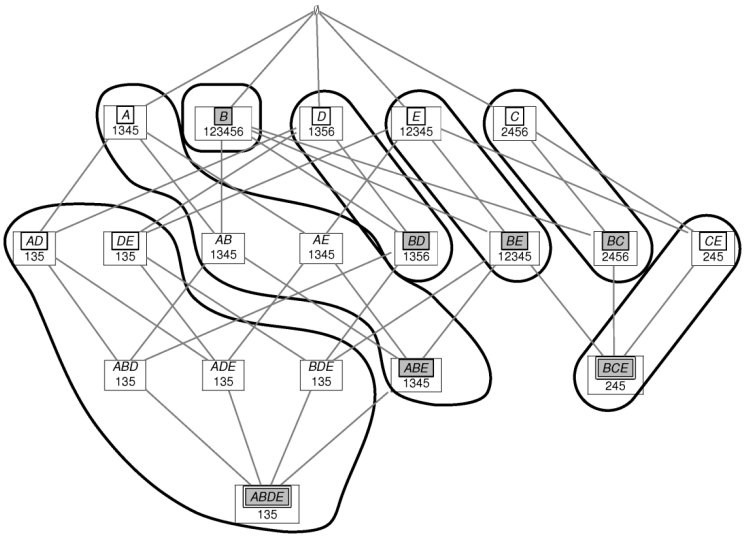
Minimal Generators

- A frequent itemset X is a minimal generator if it has no subsets with the same support

  $G = \{ X \mid X \in F \text{ and } !\exists Y \subset X, \text{ such that } \sup(X) = \sup(Y) \}$

- Thus, all subsets of X have strictly higher support, that is,

  $\sup(X) < \sup(Y)$



Shaded boxes are closed itemsets, simple boxes are generators, shaded boxes with double lines are maximal itemsets
Figure 9.2 of the textbook.

Clustering

What Is Good Clustering?

- A good clustering consists of high-quality clusters with
  - High intra-class similarity
  - Low inter-class similarity
- The quality of a clustering result depends on both
  - The similarity measure used by the method and
  - Its implementation (the algorithms used to find the clusters)
  - Its ability to discover some or all the hidden patterns
- Evaluation
  - Various measure of intra/inter cluster similarity
  - Manual inspection
  - Benchmarking on existing labels

Measure the Quality of Clustering

- Dissimilarity/Similarity metric
  - Similarity expressed in terms of distance function, typically a metric, d(i, j)
  - Definitions of distance functions are usually very different for interval-scaled, Boolean, categorical, ordinal ratio, and vector variables
  - Weights can/should be associated with different variables based on applications and data semantics
- Cluster quality measure
  - Separate from distance, there is a "quality" function that measures the "goodness" of a cluster
  - It is hard to define "similar enough" or "good enough" as the answer is typically highly subjective

Clustering Quality Measures

- Internal Validation Measures
  - Employ criteria that are derived from the data itself
  - For instance, intracluster and intercluster distances to measure cluster cohesion and separation
  - Cohesion evaluates how similar are the points in the same cluster
  - Separation, how far apart are the points in different clusters
- External Validation Measures
  - Use prior or expert-specified knowledge about the clusters
  - For example, we cluster the Iris data using the four input variables, then we evaluate the clustering using known class labels
  - Employs criteria that are not inherent to the dataset
- Relative Validation Measures
  - Aim to directly compare different solutions, usually those obtained via different parameter settings for the same algorithm.

Distance Measure

What Distance Measure?

- Euclidian distance is the typical function used to compute the similarity between two examples

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Another popular metric is city-block (Manhattan) metric, distance is the sum of absolute differences

$$d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \sum_{i=1}^{n} |x_i - y_i|$$

Jaccard Distance

- Jaccard distance is a measure of how dissimilar two sets are. Examples are represented by binary variables and are viewed as representations of set
- Jaccard distance is defined as

$$d(x, y) = 1 - J(x, y)$$

- J is the Jaccard similarity which is computed as the number of

$$J(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

- Which can also be interpreted as the percentage of identical attributes

Cosine Similarity

- The cosine similarity between x, y is simply computed as,

$$s(x,y) = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_i^n x_i^2}\sqrt{\sum_i^n y_i^2}}$$

Cosine Distance and Angular Cosine Distance

- The cosine distance between x, y is simply computed as,

$$d(x,y) = 1 - s(x,y)$$

- The angular cosine distance is computed as

$$d(x,y) = c \times \frac{\arccos(s(x,y))}{\pi}$$

with c=1 if the vectors may contain positive and negative values, c=2 if the vectors contain only positive values

Internal Measures: Cohesion and Separation

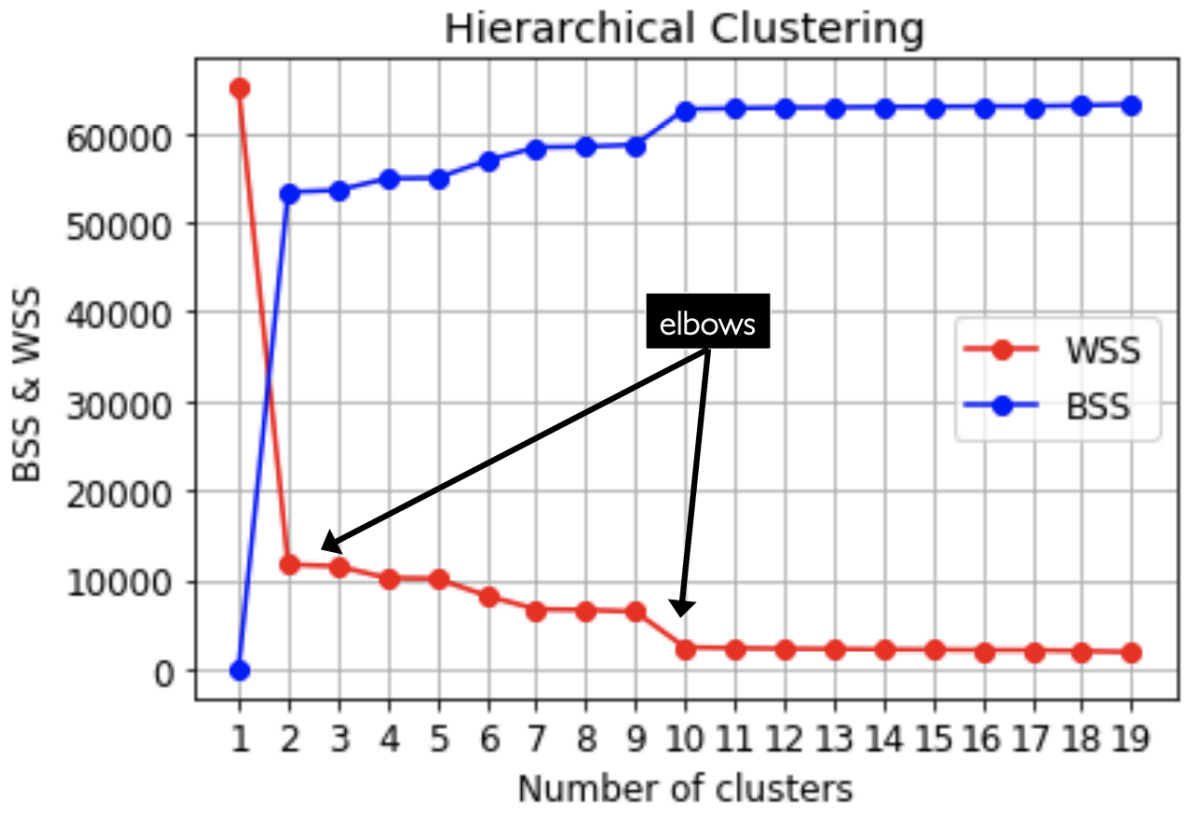
- Cohesion measures how closely related are objects in a cluster
  - Within-cluster sum of squares (μ_i is the centroid of cluster C_i if in a Euclidean space)

$$\mathrm{WSS}(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} d(x_j, \mu_i)^2$$

- Separation measures wells separated a cluster is from other clusters
  - Between-cluster sum of squares

$$\mathrm{BSS}(C) = \sum_{i=1}^{k} |C_i| d(\mu, \mu_i)^2$$

where μ is the centroid of the whole dataset



BSS and WSS for values of k from 1 up to 19.

Algorithms

Agglomerative hierarchical clustering

ALGORITHM 14.1. Agglomerative Hierarchical Clustering Algorithm

AGGLOMERATIVECLUSTERING$(\mathbf{D}, k)$:

1 $\mathcal{C} \leftarrow \{C_i = \{\mathbf{x}_i\} \mid \mathbf{x}_i \in \mathbf{D}\}$ // Each point in separate cluster
2 $\mathbf{\Delta} \leftarrow \{\delta(\mathbf{x}_i, \mathbf{x}_j) : \mathbf{x}_i, \mathbf{x}_j \in \mathbf{D}\}$ // Compute distance matrix
3 **repeat**
4 Find the closest pair of clusters $C_i, C_j \in \mathcal{C}$
5 $C_{ij} \leftarrow C_i \cup C_j$ // Merge the clusters
6 $\mathcal{C} \leftarrow \left(\mathcal{C} \setminus \{C_i, C_j\}\right) \cup \{C_{ij}\}$ // Update the clustering
7 Update distance matrix $\mathbf{\Delta}$ to reflect new clustering
8 **until** $|\mathcal{C}| = k$

Time and Space Complexity

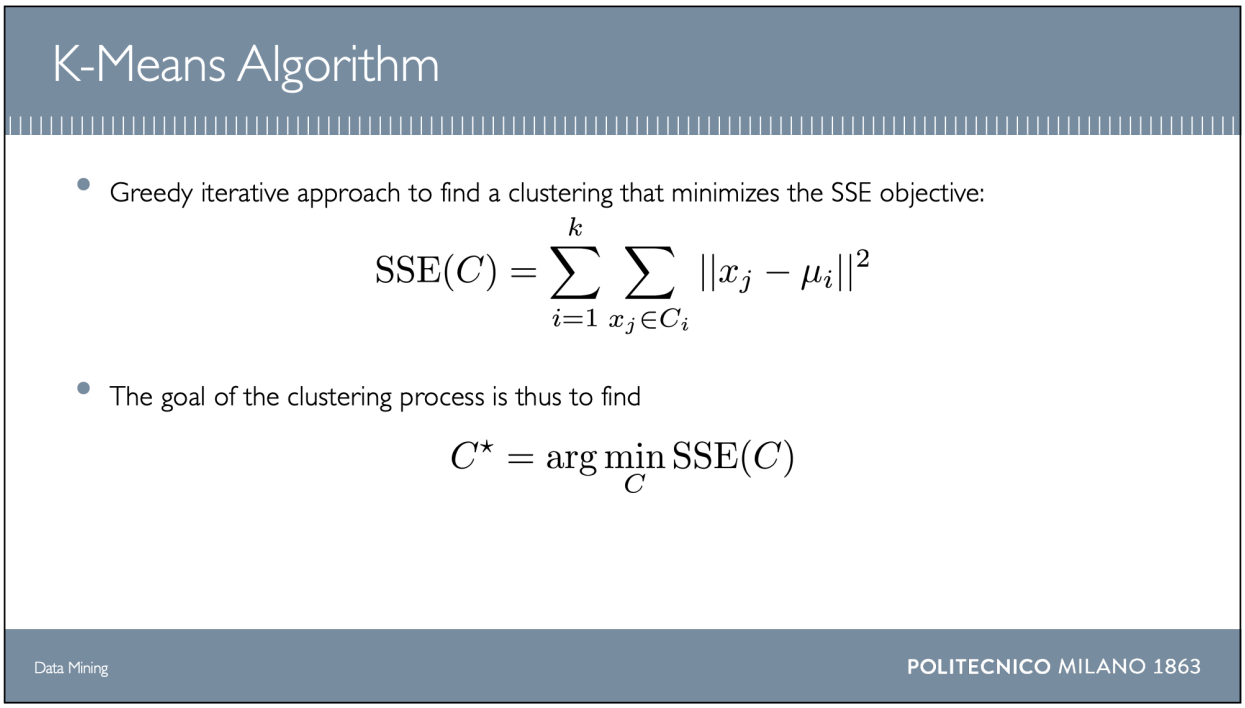
- $O(N^2)$ space since it uses the proximity matrix (N is the number of points)
- There are N steps and at each step the size, N^2, proximity matrix must be updated and searched. $O(N^3)$ time in many cases
- Complexity can be reduced to $O(N^2 \log(N))$ time for some approaches

Distance

There are 5 main methods to measure the distance between clusters, referred as linkage methods:

1. Single linkage: computes the minimum distance between clusters before merging them.
2. Complete linkage: computes the maximum distance between clusters before merging them.
3. Average linkage: computes the average distance between clusters before merging them.
4. Centroid linkage: calculates centroids for both clusters, then computes the distance between the two before merging them.
5. Ward's (minimum variance) criterion: minimizes the total within-cluster variance and finds the pair of clusters that leads to minimum increase in total within-cluster variance after merging.

K-Means algorithm



Algorithm 13.1: K-means Algorithm

K-MEANS $(\mathbf{D}, k, \epsilon)$**:**

$t = 0$

Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$

repeat

  $t \leftarrow t + 1$

  $C_i \leftarrow \emptyset$ for all $i = 1, \cdots, k$

  `// Cluster Assignment Step`

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    $i^* \leftarrow \arg\min_i \left\{ \|\mathbf{x}_j - \boldsymbol{\mu}_i^{t-1}\|^2 \right\}$

    $C_{i^*} \leftarrow C_{i^*} \cup \{\mathbf{x}_j\}$ `// Assign` $\mathbf{x}_j$ `to closest centroid`

  `// Centroid Update Step`

  foreach $i = 1, \cdots, k$ **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$

until $\sum_{i=1}^{k} \|\boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1}\|^2 \leq \epsilon$

Computational Complexity

- Cluster assignment takes O(nkd) time since, for each of the n points, it computes its distance to each of the k clusters, which takes d operations in d dimensions
- The centroid re-computation step takes O(nd) time to add n d-dimensional points
- Assuming that there are t iterations, the total time for K-means is given as O(tnkd).
- In terms of the I/O cost it requires O(t) full database scans, because we have to read the entire database in each iteration.

Algorithm 3 Bisecting K-means Algorithm.

1: Initialize the list of clusters to contain the cluster containing all points.
2: **repeat**
3: Select a cluster from the list of clusters
4: **for** $i = 1$ to *number_of_iterations* **do**
5: Bisect the selected cluster using basic K-means
6: **end for**
7: Add the two clusters from the bisection with the lowest SSE to the list of clusters.
8: **until** Until the list of clusters contains K clusters

Limitations of K-means

- K-means has problems when clusters are of differing
  - Sizes
  - Densities
  - Non-globular shapes
- K-means has also problems when the data contains outliers

k-Means Clustering Summary

- Strength
  - Relatively efficient
  - Often terminates at a local optimum
  - The global optimum may be found using techniques such as: deterministic annealing and genetic algorithms
- Weakness
  - Applicable only when mean is defined, then what about categorical data?
  - Need to specify k, the number of clusters, in advance
  - Unable to handle noisy data and outliers
  - Not suitable to discover clusters with non-convex shapes
- Advantages
  - Simple, understandable
  - Items automatically assigned to clusters
- Disadvantages
  - Must pick number of clusters before hand
  - All items forced into a cluster
  - Too sensitive to outliers

Mean Shift Algorithm Pseudocode

1. Choose a search window size (the bandwidth)
2. Choose the initial location of the search window
3. Compute the mean location in the search window
4. Center the search window at the location computed in Step 3
5. Repeat Steps 3 and 4 until convergence

Mean Shift Clustering - Pseudocode

1. Choose kernel and bandwidth
2. For each point
   a. Center a window on that point
   b. Compute the mean of the data in the search window
   c. Center the search window at the new mean location
   d. Repeat (b,c) until convergence
3. Assign points that lead to nearby modes to the same cluster

K-Means & Mean Shift Clustering

- Number of clusters and shape
  - k-means assumes that the number of clusters is already known and the clusters are shaped spherically
  - Mean shift does not assume anything about number of clusters.
  - Instead, the number of modes give the number of clusters
  - Also, since mean shift is based on density estimation, it can handle arbitrarily shaped clusters.
- Initialization and outliers
  - k-means is sensitive to initialization. Mean shift is robust to initializations.
  - Typically, mean shift is run for each point or sometimes points are selected uniformly from the feature space
  - Similarly, k-means is sensitive to outliers while Mean Shift is less sensitive.
- Time complexity
  - k-means is fast and has a time complexity O(knT) where k is the number of clusters, n is the number of points and T is the number of iterations.
  - Classic mean shift is computationally expensive witha time complexity $O(Tn^2)$.

Mean Shift Clustering

- Strength
  - Does not assume any prior shape on data clusters
  - Can handle arbitrary feature spaces
  - Only the window size h to choose
  - The window size h (or bandwidth) has a physical meaning
- Weaknesses
  - The window size h (or bandwidth) selection is not trivial. Inappropriate window size can cause modes to be merged, or generate additional "shallow" modes
  - Adaptive window size can help
  - Not suitable for high-dimensional features

Expectation-Maximization (EM) Clustering

- k-means assigns each point to only one cluster (hard assignment)
- The approach can be extended to consider soft assignment of points to clusters, so that each point has a probability of belonging to each cluster
- We assume that each cluster C_i is characterized by a multivariate normal distribution and thus identified by the mean vector $\boldsymbol{\mu}_i$ and covariance matrix $\boldsymbol{\Sigma}_i$
- A clustering is identified by a vector of parameter $\boldsymbol{\theta}$ defined as

  $\boldsymbol{\theta} = \{\boldsymbol{\mu}_i\ \boldsymbol{\Sigma}_i\ P(C_i)\}$

  where $P(C_i)$ are the prior probability of all the clusters C_i which sum up to one

Expectation-Maximization (EM) Clustering

- The goal of maximum likelihood estimation (MLE) is to choose the parameters $\boldsymbol{\theta}$ that maximize the likelihood, that is

$$\theta^\star = \arg\max_\theta P(D|\theta)$$

- General idea
  - Starts with an initial estimate of the parameter vector
  - Iteratively rescores the patterns against the mixture density produced by the parameter vector
  - The rescored patterns are used to update the parameter updates
  - Patterns belonging to the same cluster, if they are placed by their scores in a particular component

DBSCAN: Basic Concepts

- The neighborhood within a radius ε of a given object is called the ε-neighborhood of the object

$$N_\epsilon(x) = \{y | \delta(x, y) \leq \epsilon\}$$

- A Core Point contains at least minpts objects in its ε-neighborhood
- A Border Point is not a core point, but it is inside the neighborhood of a core point
- A Noise Point is not a core point nor a border point

Complexity of DBScan

- DBSCAN needs to compute the ε-neighborhood for each point
- If the dimensionality is not too high this can be done efficiently using a spatial index structure in O(n log n)
- If the dimensionality is high, it takes $O(n^2)$ to compute the neighborhood for each point.
- Once the ε-neighborhood has been computed the algorithm needs only a single pass over all the points to find the density connected clusters.
- The overall complexity of DBSCAN is between O(n log n) and $O(n^2)$ in the worst-case.

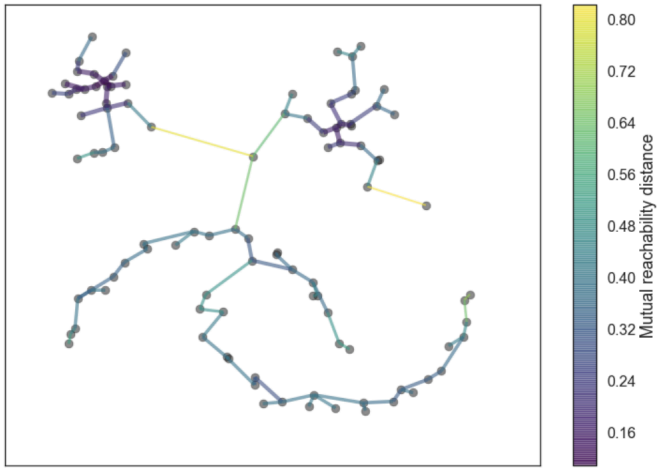
DBScan fails when applied to data of varying density

HDBSCAN

- Extends DBSCAN by converting it into a hierarchical clustering algorithm, and then using a technique to extract a flat clustering based in the stability of clusters
- It defines the "core distance" of a point as the minimum distance of a point to its k-th nearest neighbor or $core_k(x)$
- Thus, points in high density areas will have a low core distance, while point in low density areas will have a high core distance
- This provides a local inexpensive estimate of density

Next, Build the Minimum Spanning Tree

- Consider the data as a weighted graph
- Data points are vertices. An edge between any two points with weight equal to the mutual reachability distance of those points
- Build the spanning tree using Prim's algorithm one edge at a time by adding the lowest weight edge that connects the current tree to a vertex not yet in the tree

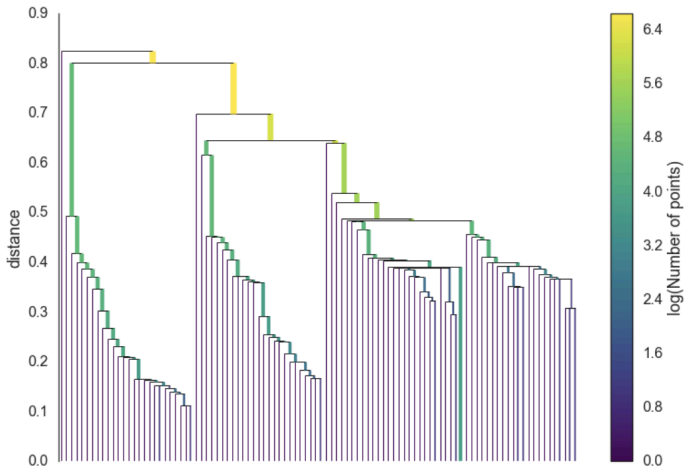


https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

HDBSCAN converts the minimum spanning tree into a hierarchy of connected components

It sorts the edges of the tree by distance (in increasing order)

Then iterate creating a new merged cluster for each edge.



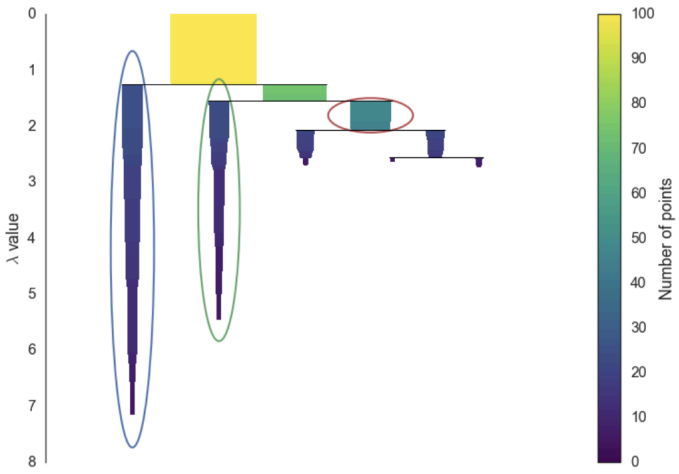
https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

Condensing the Tree

- We introduce the notion of "minimum cluster size" threshold
- We navigate the hierarchy and from the top and at each split we check the size of the merged clusters
- If a clusters has fewer points that the threshold, then the smaller clusters is eliminated (the points have "fallen out of the cluster")
- Otherwise both clusters are maintained.
- Note that, which a point p 'fell out of the cluster' and at what distance value that happened is stored as $\lambda_p = 1/\text{distance}$
- At the end, a much smaller tree with a small number of nodes remains with node has data about how the size of the cluster at that node decreases over varying distance

Extracting the Cluster

- Given the simplified dendrogram we want to select clusters that persist and have a longer lifetime since short lived clusters probably represent artifacts
- We may say that we want to choose those clusters that have the greatest area of ink in the plot. Also, we cannot select any a descendant of a cluster we chose.



By applying the two principles (larger total ink area and no descendents) we extract three clusters
https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

Linear Regression

The goal of linear regression is thus to find the weights that minimize RSS

$$w_0, w_1 = \arg\min_{w_0,w_1} RSS(w_0, w_1)$$

$$= \arg\min_{w_0,w_1} \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i))^2$$

Coefficient of Determination R^2

- Total sum of squares

$$TSS = \sum_{i=1}^{N} (y_i - \bar{y})^2$$

- Coefficient of determination

$$R^2 = 1 - \frac{RSS}{TSS}$$

- R^2 measures of how well the regression line approximates the real data points. When R^2 is 1, the regression line perfectly fits the data.

Holdout Evaluation

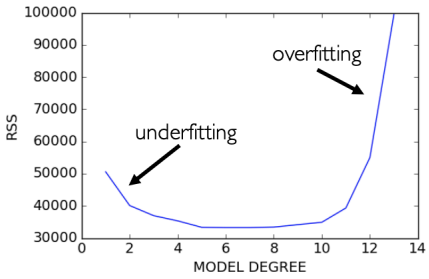
- Reserves a certain amount for testing and uses the remainder for training
  - Too small training sets might result in poor weight estimation
  - Too small test sets might result in a poor estimation of future performance
- Typically,
  - Reserve ½ for training and ½ for testing
  - Reserve 2/3 for training and 1/3 for testing
- For small or "unbalanced" datasets, samples might not be representative

Cross-Validation

- First step
  - Data is split into k subsets of equal size
- Second step
  - Each subset in turn is used for testing and the remainder for training
- This is called k-fold cross-validation and avoids overlapping test sets
- Often the subsets are stratified before cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

Underfitting vs Overfitting

- Having too few parameters leads to underfitting
  - Model isn't powerful enough to describe data
- While too many parameters leads to overfitting
  - Model is too powerful
- By increasing degree of polynomial
  - First see improvements in cross-validation error
  - Then error starts to increase again as model starts to overfit
- At the end, best performance for 5th degree polynomial



Lasso Regression (L_1 Regularization)

- Minimizes the cost function,

$$\begin{aligned} Cost(\vec{w}) &= RSS(\vec{w}) + \alpha||\vec{w}||_1 \\ &= \sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{D} w_j h_j(x_i)\right)^2 + \alpha\sum_{j=0}^{D}|w_j| \end{aligned}$$

- If α is zero, the cost is exactly the same as before; if α is infinite, then the only solution corresponds to having all the weights to 0. In gradient descent, weight j is modified as,

$$z_j = \sum_{i=1}^{N} h_j(\vec{x_i})^2$$

$$\rho_j = \sum_{i=1}^{N} h_j(\vec{x_i})(y_i - \hat{y}_i(\vec{w}_{-j}^{(t)}))$$

$$w_j = \begin{cases} (\rho_j + \alpha/2)/z_j & \texttt{if } \rho_j < -\alpha/2 \\ 0 & \texttt{if } \rho_j \in [-\alpha/2, \alpha/2] \\ (\rho_j - \alpha/2)/z_j & \texttt{if } \rho_j > \alpha/2 \end{cases}$$

Ridge Regression (L_2 Regularization)

- Minimizes the cost function,

$$Cost(\vec{w}) = RSS(\vec{w}) + \alpha ||\vec{w}||_2^2$$
$$= \sum_{i=1}^{N} \left(y_i - \sum_{j=0}^{D} w_j h_j(x_i) \right)^2 + \alpha \sum_{j=0}^{D} w_j^2$$

- If α is zero, the cost is exactly the same as before; if α is infinite, then the only solution corresponds to having all the weights to 0

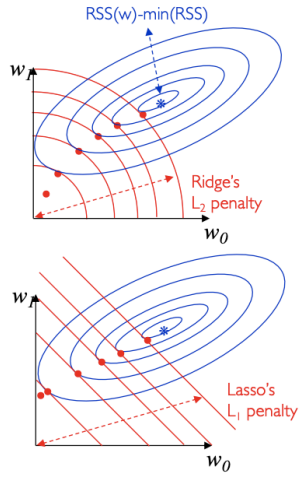
- In the gradient descent algorithm, the update for weight j becomes,

$$\Delta w_j = -2 \sum_{i=1}^{N} h_j(\vec{x}_i)(y_i - \hat{y}_i(\vec{w}^{(t)})) + 2\alpha w_j$$

Explaining Lasso's Sparsity Effect

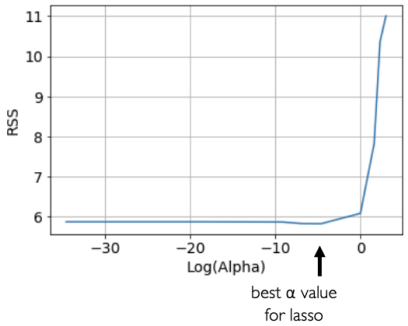
- Graphs show level sets (lines of equal value) for
  - RSS loss function in blue and
  - Ridge and Lasso penalties in red

- As we increase alpha
  - Optimal solution to penalized loss function moves back toward the origin (red dots)
  - For lasso it first heads to an axis, removing the weight altogether
  - For ridge, the optimal point never reaches the axis



Selecting the Best α

- To select best value of α
  - cannot use test set since it will be used for evaluating final model (which uses α)
  - need to reserve part of training data to evaluate candidates and select best value
- If have enough data, extract validation set from training data to select α
- If don't have enough data
  - Select α by applying k-fold cross-validation over training data
  - Choose α corresponding to lowest average cost over k folds



Introduction to Classification

Classification

The target to predict is a label (the class variable)
(good/bad, none/soft/hard, etc.)

Regression

The target to predict is numerical

Logistic Regression

Intuition Behind Linear Classifiers (Two Classes)

- We define a score function similar to the one used in linear regression,

$$Score(\vec{x_i}) = \sum_{j=0}^{D} w_j h_j(\vec{x_i})$$

- The label is determined by the sign of the score value,

$$\hat{y}_i = sign(Score(\vec{x_i}))$$
$$= \begin{cases} +1 & \texttt{if } Score(\vec{x_i}) \geq 0 \\ -1 & \texttt{if } Score(\vec{x_i}) < 0 \end{cases}$$

Logistic Regression

- Well-known and widely used statistical classification method. It computes the probability of assigning a class to an example, that is,

$$P(y_i|\vec{x_i})$$

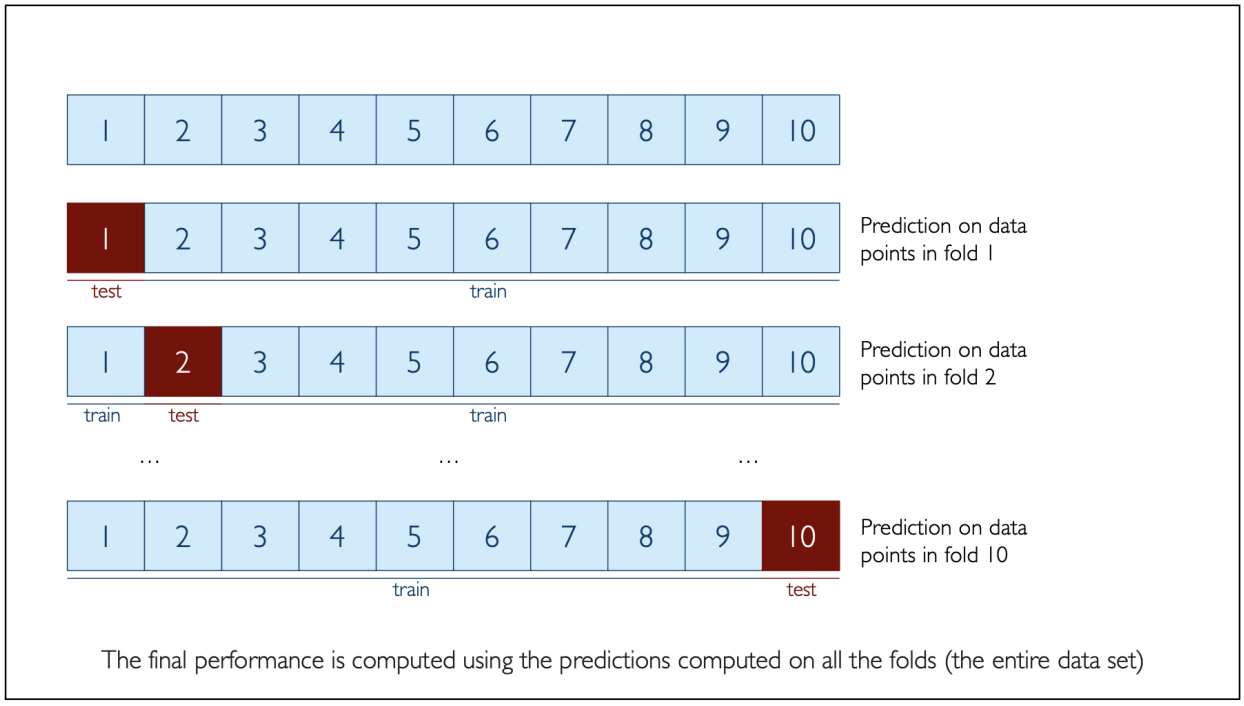
- For this purpose, logistic regression assumes that,

$$P(\hat{y}_i = +1|\vec{x_i}) = \frac{1}{1 + e^{-Score(\vec{x_i})}}$$

- If we use h to identify all the feature transformation h_j we get,

$$P(\hat{y}_i = +1|\vec{x_i}, \vec{w}) = \frac{1}{1 + e^{-\vec{w}h(\vec{x_i})}}$$

Cross Validation



The final performance is computed using the predictions computed on all the folds (the entire data set)

From now on we are going to use k-fold cross-validation a lot to score models

k-fold cross-validation generates k separate models

Which one should be deployed?

K-fold cross-validation provides an evaluation of model performance on unknown data

Its output it's the evaluation, not the model!

The model should be obtained by running the algorithm on the entire dataset!

Classification metrics

The Confusion Matrix

- Focus on the predictive capability of a model
- Confusion Matrix:

		PREDICTED CLASS	
		Yes	No
TRUE CLASS	Yes	TP true positives	FN false negatives
	No	FP false positives	TN true negatives

Metrics for Performance Evaluation: Accuracy

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	#TP	#FN
	No	#FP	#TN

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision & Recall

- Alternatives to accuracy, introduced in the area of information retrieval and search engine
- Precision
  - Percentage of items classified as positive that are actually positive
  - In the information retrieval context represents the percentage of actually good documents that have been shown as a result.
- Recall
  - Percentage of positive examples that are classified as positive
  - In the information retrieval context, recall represents the percentage of good documents shown with respect to the existing ones.

Cost-Sensitive Measures

$$Precision(p) = \frac{TP}{TP+FP} = \frac{a}{a+c}$$

The higher the precision, the lower the FPs

$$Recall(r) = \frac{TP}{TP+FN} = \frac{a}{a+b}$$

The higher the recall, the lower the FNs

$$F1-measure = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

The higher the F1, the lower the FPs & FNs

- Precision is biased towards TP) & FP
- Recall is biased towards TP & FN
- F1-measure is biased towards all except TN it is high when both precision and recall are reasonably high

Sensitivity & Specificity

- Sensitivity evaluates the ability to correctly identify the elements of the positive class and it is computed as the true positive rate (TPR)

  TPR = TP/(TP+FN)

- Specificity estimates the probability to correctly identify the elements of the negative class and it is computed as the true negative rate

  TNR = TN/(TN+FP)

How do we know that the difference
in performance is not just due to chance?

We computes the odds of it!

Apply the t-test and compute the p-value

The p-value represents the probability
that the reported difference is due to chance

Bonferroni Correction

- Assume that individual tests are independent.
- Divide the desired p-value threshold by the number of tests performed.
- Example
  - We now have, the threshold set to 0.05/20 = 0.0025.
  - P(making a mistake) = 0.0025
  - P(not making a mistake) = 0.9975
  - P(not making any mistake) = 0.9975^{20} = 0.9512
  - P(making at least one mistake) = 1 - 0.9512 = 0.0488

Probabilistic classifiers

Probabilistic Classifiers & Classification Thresholds

- Up to now we used logistic regression to predict classifier labels, however, logistic regression returns a probability

$$P(y_i|\vec{x_i})$$

- Given an example x_i its predicted class is the label with the largest probability, so it is equivalent to using a threshold of 0.5 to decide which class to assign to an example
- However, we can use a different threshold and for instance label as positive only examples we return a 1 only when P(+1|x)>0.75
- This would label as positive only cases for which we are more confident that should be labeled as positive.

We can use the threshold to
optimize our precision and recall

a higher the threshold,
increases precision and lower recall

a lower threshold, decreases
precision and increase recall

ROC Curves

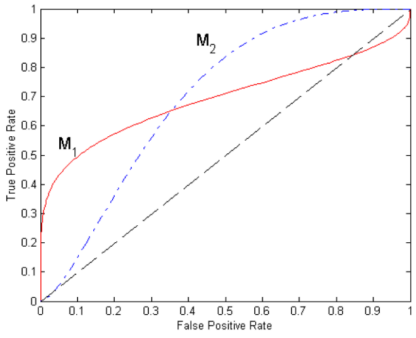
Receiver Operating Characteristic (ROC) Curve

- Developed in 1950s for signal detection theory to analyze signals
- Plot the True Positive Rate (TPR=TP/(TP+FN))
  against the False Positive Rate (FPR=FP/(TN+FP))
- Performance of each classifier represented as a point on the ROC curve
- By changing the classification threshold, the cost matrix changes the location of the point

Using ROC for Model Comparison

- No model consistently outperform the other
- M_1 is better for small FPR
- M_2 is better for large FPR

- Area Under the ROC curve
- Ideal, area = 1
- Random guess, area = 0.5



How to Compute an ROC Curve

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

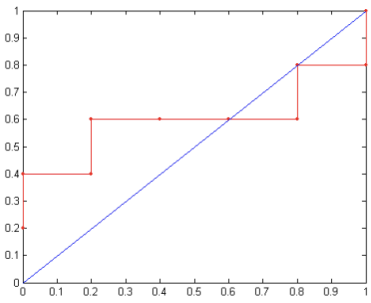
- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/(TP+FN)
- FP rate, FPR = FP/(FP + TN)

How to Compute an ROC Curve

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Threshold = 1.0
  - None is classified as +
  - TP = 0, TN = 5, FN = 5
  - TPR = 0 FPR = 0
- Threshold = 0.95
  - One is classified as +
  - TP = 1, TN = 5, FN = 4
  - TPR = 0.2 FPR = 0
- Threshold = 0.93
  - Two + classified as +
  - TP = 2, TN = 5, FN = 3
  - TPR = 0.4 FPR = 0

Class	+	-	+	-	-	-	+	-	+	+	
P	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0



Naïve Bayes and Bayesian Networks

Naïve Bayes for Classification

- Training
  - Count the frequency of tuples (x_i, y) for each attribute value x_i and each class value y
  - Use the counts to compute estimates for the class probability P(y) and the conditional probability $P(x_i|y)$
- Testing
  - Given an example x, computes the most likely class as

$$\begin{aligned} class &= \arg\max_y P(y|\vec{x}) \\ &= \arg\max_y \frac{P(x_1|y)\cdots P(x_n|y)P(y)}{P(\vec{x})} \\ &= \arg\max_y P(x_1|y)\cdots P(x_n|y)P(y) \end{aligned}$$

Using the Probabilities for Classification

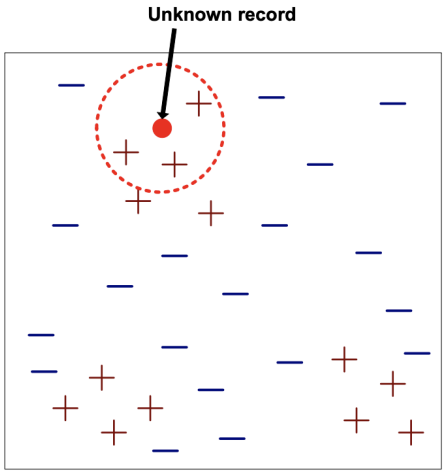
Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

$$\begin{aligned} \text{Likelyhood of yes} &= P(Sunny|yes)P(Cool|yes)P(High|yes)P(True|yes)P(yes) \\ &= 0.0053 \\ \text{Likelyhood of no} &= P(Sunny|no)P(Cool|no)P(High|no)P(True|no)P(no) \\ &= 0.0206 \end{aligned}$$

- The sum of the two values should be one. Thus, we convert the two values into actual probabilities using normalization:

$$\begin{aligned} P(yes|Sunny, Cool, High, True) &= 0.0053/(0.0053+0.0206) = 0.205 \\ P(no|Sunny, Cool, High, True) &= 0.0206/(0.0053+0.0206) = 0.795 \end{aligned}$$

k-Nearest Neighbors



Three elements

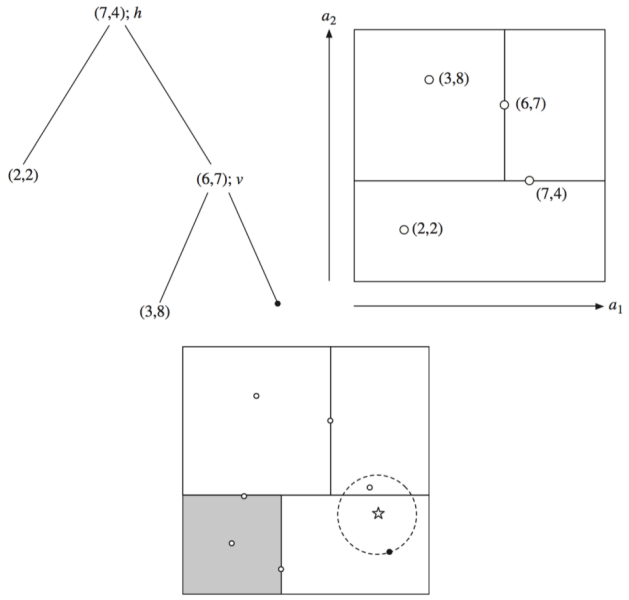
- The training dataset
- Similarity function (or distance metric)
- The number of nearest neighbors to retrieve k

Classification

- Compute distance to other training records
- Identify the k nearest neighbors
- Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Finding Nearest Neighbors Efficiently

- Basic Approach
  - Linear scan of the data
  - Classification time for a single distance depends on the number of data points and the number of variables O(nd) for n instances of d variables
  - This becomes prohibitive when the training set is large
- Nearest-neighbor search can be speeded up by using
  - KD-Trees
  - Ball-Trees
  - …



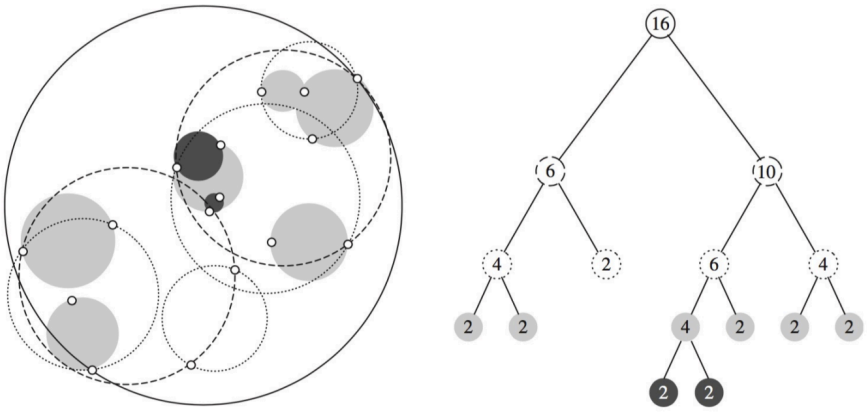
Example of KD-tree. The tree works like a decision tree and splits the space in a hierarchy. When the class of one example is needed, the tree is navigated until a leaf is reached and only the nearby areas are actually checked by backtracking on the tree.

Effectiveness of KD-trees

- Search complexity depends on depth of tree.
- It is the logarithm of number of nodes for balanced tree O(log(n))
- Occasional rebalancing of tree may be needed randomizing order of data is another option
- But amount of backtracking required depends on quality of tree
- Some nodes are square (good) while others are skinny (bad)

Ball Trees

- Corners in high-dimensional space may mean query ball intersects with many regions and this is a potential limitation for KD-Tree
- Note that there is no need to make sure that regions do not overlap, so they do not heed to be hyper rectangles
- Can use hyper spheres (balls) instead of hyper rectangles
- A ball tree organizes the data into a tree of k-dimensional hyperspheres
- Balls may allow for a better fit to the data and thus more efficient search



Decision Trees

Which Attribute for Splitting?

- At each node, available attributes are evaluated based on separating the classes of the training examples using either a purity or impurity measure
- Typical measures used are the information gain (ID3), information gain ratio (C4.5), gini index (CART)
- Information Gain increases with the average purity of the subsets that an attribute produces. It selects the attribute with the highest information gain

Computing Information

- Information is measured in bits
  - Given a probability distribution, the info required to predict an event is the distribution's entropy
  - Entropy gives the information required in bits (this can involve fractions of bits!)
- Formula for computing the entropy

$$entropy(p_1, p_2, \ldots, p_n) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \cdots - p_n \log_2 p_n$$

- Note that, since entropy refers to the amount of information expressed in bits, it uses the logarithm with base 2

Information Gain

Information Gain (IG)

- Difference between the information before split and the information after split

$$gain(A) = info(D) - info_A(D)$$

- The information before the split is the entropy of the class distribution in D,

$$info(D) = -p_1 \log_2 p_1 \cdots - p_n \log_2 p_n$$

- The information after the split using attribute A is computed as the weighted sum of the entropies on each split, given n splits,

$$info_A(D) = \frac{|D_1|}{|D|} info(D_1) + \cdots + \frac{|D_n|}{|D|} info(D_n)$$

Information Gain Ratio

Information Gain Ratio (IGR)

- Modification of the Information Gain that reduces the bias toward highly-branching attributes
- Information Gain Ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Information Gain Ratio takes number and size of branches into account when choosing an attribute
- It corrects the information gain by taking the intrinsic information of a split into account

Intrinsic Information (IR)

- Intrinsic information

$$IntrinsicInfo(S, A) = -\sum \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|}$$

  computes the entropy of distribution of instances into branches

- Intrinsic Information is the entropy of A, independent of the class.

- Information Gain Ratio normalizes Information Gain by

$$GainRatio(S, A) = \frac{Gain(S, A)}{IntrinsicInfo(S, A)}$$

The Gini Index

The Gini Index

- The Gini index, for a data set T contains examples from n classes, is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

  where pj is the relative frequency of class j in T

- gini(T) is minimized if the classes in T are skewed

- gini measures error rate of random classifier that assigns classes to instances according to their prior frequencies:

$$gini(T) = \sum_j p_j(1 - p_j) = \sum_j p_j(1 - p_j) = \sum_j p_j - p_j^2 = 1 - \sum_j p_j^2$$

The Gini Index

- If a data set D is split on A into two subsets D_1 and D_2 then,

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- The reduction of impurity is defined as,

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest Gini splitting D over A (or the largest reduction in impurity) is chosen to split the node (need to enumerate all the possible splitting points for each attribute)

Pre/Post-Pruning

Pre-pruning vs Post-pruning

- Prepruning
  - Halt tree construction early
  - Do not split a node if this would result in the goodness measure falling below a threshold
  - Difficult to choose an appropriate threshold
- Postpruning
  - Remove branches from a "fully grown" tree
  - Get a sequence of progressively pruned trees
  - Use a set of data different from the training data to decide which is the "best pruned tree"

Prepruning

- Based on statistical significance test
- Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- The most popular approach is the chi-squared test
- Quinlan's classic tree learner ID3 used chi-squared test in addition to information gain
- Only statistically significant attributes were allowed to be selected by the information gain procedure

Post-Pruning

- First, build full tree, then prune it
- Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations
  - Subtree raising
  - Subtree replacement
- Possible strategies
  - Error estimation
  - Significance testing
  - MDL principle

C4.5's Pruning Method

- Given the error f on the training data, the upper bound for the error estimate for a node is computed as

$$e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Big/ \left(1 + \frac{z^2}{N} \right)$$

- If c = 25% then z = 0.69
  - f is the error on the training data
  - N is the number of instances covered by the leaf

Regression & Model Trees

- Regression Trees
  - Prediction is computed as the average of numerical target variable in the subspace
- Model Trees
  - Leaves use a linear model to predict the target value in the subspace

- What impurity measure?
  - It can be measured as the expected error reduction, or SDR (standard deviation reduction)

$$SDR = \sigma(D) - \sum_i \frac{|D_i|}{|D|}\sigma(D_i)$$

  - Where D is the original data D_i are the partitions and **σ** is the standard deviation of the target

Ensemble Methods

Ensemble Methods

Generate a set of classifiers from the training data

Predict class label of previously unseen cases by aggregating predictions made by multiple classifiers

Majority vote for classification
Averaging for regression

Building models ensembles

- Basic idea
  - Build different "experts", let them vote
- Advantage
  - Often improves predictive performance
- Disadvantage
  - Usually produces output that is very hard to analyze

Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate, ε = 0.35
- Assume classifiers are independent
- The probability that the ensemble makes a wrong prediction is

$$P(X \geq 13 \mid p = \epsilon, n = 25) = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

Bagging (Bootstrap Aggregation)

What is Bagging? (Bootstrap Aggregation)

- Training
  - Given a set D of d tuples, generate k training dataset D_i using bootstrap
  - Compute k models M_i using the training sets D_i
- Predicting (classify an unknown sample x)
  - Each classifier M_i computes its prediction for x
  - The bagged classifier M* return the class predicted by the majority of the models
  - When class values are -1 and 1, the output of the ensemble can be computed as

$$M^*(x) = sign\left(\sum_{t=1}^{k} M_i(x)\right)$$

Random Forests

What is a Random Forest?

- Random forests (RF) are a combination of tree predictors
- Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest
- The generalization error of a forest of tree classifiers depends on
  - The strength of the individual trees in the forest (how good are they on average?)
  - The correlation between them (how much group-think is there in the predictions)
- Using a random selection of features to split each node yields error rates that are more robust with respect to noise

Out of Bag Samples

- For each observation (x_i, y_i), construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which the observation did not appear
- The OOB error estimate is almost identical to that obtained by n-fold crossvalidation and related to the leave-one-out evaluation
- Thus, random forests can be fit in one sequence, with cross-validation being performed along the way
- Once the OOB error stabilizes, the training can be terminated

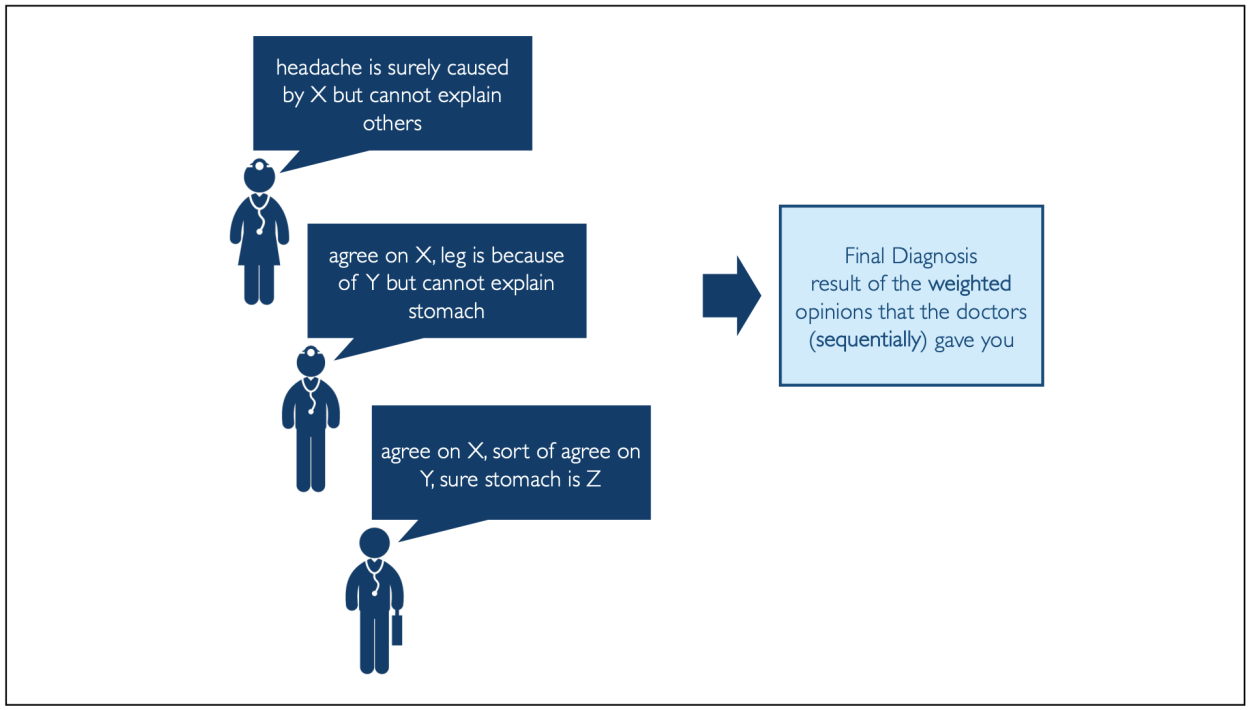
OOB Errors for Random Forests

The `RandomForestClassifier` is trained using *bootstrap aggregation*, where each new tree is fit from a bootstrap sample of the training observations $z_i = (x_i, y_i)$. The *out-of-bag* (OOB) error is the average error for each z_i calculated using predictions from the trees that do not contain z_i in their respective bootstrap sample. This allows the `RandomForestClassifier` to be fit and validated whilst being trained [1].

Properties of Random Forests

- Easy to use ("off-the-shelve"), only 2 parameters (no. of trees, %variables for split)
- Very high accuracy
- No overfitting if selecting large number of trees
- Insensitive to the choice of split% (~20%)
- Returns an estimate of variable importance

Boosting



How Does Boosting Work?

- Weights are assigned to each training example
- A series of k classifiers is iteratively learned
- After a classifier M_i is learned, the weights are updated
- The next classifier M_{i+1} will focus more on the training tuples that were misclassified by M_i
- The final M* is a weighted sum of all the classifiers' outputs

AdaBoost

- AdaBoost computes a strong classifier as a combination of weak classifiers,

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- Where $h_t(x)$ is the output of the t^{th} weak classifier t
- α_t is weight assigned to the t^{th} weak classifier based on its estimated error ϵ_t as

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

- Note that this "confidence" weight is just the log odds of a correct prediction!

Gradient Tree Boosting

- Build a sequence of tree predictors by repeating three simple steps
  1. Learn a basic predictor
  2. Compute the gradient of a loss function with respect to the predictor
  3. Compute a model to predict the residual
  4. Updated the predictor with the new model
  5. Goto 2
- The predictor is increasingly accurate and increasingly complex

Why is it Called Gradient Boosting?

- Intuition using MSE: we want to compute a model to predict a target value y_i that minimizes a loss function, for instance the mean square error

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- We can adjust $\hat{y}_i$ to try to reduce the error using the gradient

$$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$

- The gradient for the MSE is proportional to $y_i - \hat{y}_i$
- Thus, each learner is estimating the gradient of the loss. Larger α means larger steps, smaller α smaller steps and smoothing effect

eXtreme Gradient Boosting (XGBoost)

- Efficient and scalable implementation of gradient boosting for classification and regression trees
- Used by most of the winning solutions and runner up of the recent data science competitions
- It only deals with numerical variables

- Features
  - Novel tree learning algorithm to handle sparse data
  - Approximate tree learning using quantile sketch
  - More than ten times faster on single machines
  - More regularized model formalization to control over-fitting, which gives it better performance.

Light Gradient Boosting Machine (LightGBM)

- Open source distributed gradient boosting originally developed by Microsoft that supports different algorithms (including Gradient Boosting Trees and Random Forests)
- It has many of XGBoost's advantages, including sparse optimization, parallel training, multiple loss functions, regularization, bagging, and early stopping.
- The major difference lies in the construction of trees. LightGBM grows trees leaf-wise and chooses the leaf it estimates will yield the largest decrease in loss

- Features
  - Faster training speed and higher efficiency
  - Lower memory usage
  - Better accuracy
  - Support of parallel, distributed, and GPU learning
  - Capable of handling large-scale data

Stacked generalization (Stacking)

Stacking Generalization (Stacking)

- Train a learning algorithm to combine the predictions of a set of heterogeneous models
  - First, a set of base learners are trained over the data (to generate level-0 models)
  - Then, a meta learner is trained using the predictions of base classifiers as additional inputs (to generate a level-1 model)
- Level-0 models are generated different learning schemes
- Typically yields performance better than any single one of the trained models

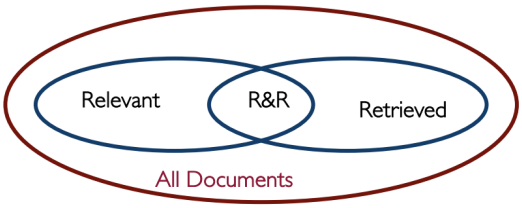
Text Representation and Embeddings

Two Modes of Text Access

- Pull Mode (search engines)
  - Users take initiative
  - Ad hoc information need
- Push Mode (recommender systems)
  - Systems take initiative
  - Stable information need or system has good knowledge about a user's need

Text Retrieval Methods

- Document Selection (keyword-based retrieval)
  - Query defines a set of requisites
  - Only the documents that satisfy the query are returned
  - A typical approach is the Boolean Retrieval Model
- Document Ranking (similarity-based retrieval)
  - Documents are ranked on the basis of their relevance with respect to the user query
  - For each document a "degree of relevance" is computed with respect to the query
  - A typical approach is the Vector Space Model



$$\text{precision} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{\{\text{Retrieved}\}}$$

$$\text{recall} = \frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{\{\text{Relevant}\}}$$

$$\text{F-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$F_{score} = \frac{recall \times precision}{(recall + precision)/2}$$

Document Representation: Term Frequency (TF)

- To take into account the number of times a word appears in the document di and qi now represent the number of times word wi appears in the document.
- Note that, this is a simplification for the discussion. In fact, we will actually use the relative frequency of the words inside the documents when applying it.

Inverse Document Frequency (IDF)

- It measures how much information the word provides (how much it is surprising to have it in the document).
- Penalizes the words that frequently occur in many documents,

$$IDF(w) = \log \frac{M}{k}$$

- Where k is the number of documents in which w appears M is the number of documents.
- When w might is not in the corpus, the formula leads to a division-by-zero so the adjusted version with (k+1) at the denominator is used.

Document Similarity

- We combine Term Frequency (TF) with Inverse Document Frequency to build a representation of documents that we can use to compute document similarity or distance. Given a document d from a corpus containing M documents, the vector elements d_i are computed as,

$$d_i = tf(w_i, d) \log \frac{M}{k_i}$$

- Where k_i is the number of documents that contain w_i while tf(w_i,d) is the frequency of word w_i in document d. Note that in the examples, we always used word counts c(w,d) since the values are easier to manage, but term frequency tf is used in practice.
- We can use the vectors to compute distance (e.g., using cosine distance or Euclidean distance for example) or similarity (using cosine similarity)

Blackbox Optimization & Evolutionary Computation

Initialization	Randomly generates the initial population of candidate solutions (strings, real vectors).
Evaluation	Evaluates the population of candidate solutions using the given fitness function.
Selection	Selects promising solutions from the current population by making more copies of better solutions at the expense of the worse ones.
Variation	Processes selected solutions to generate new candidate solutions that share similarities with selected solutions but are novel in some way.
Replacement	Incorporates new candidate solutions into the original population.

A Simple Genetic Algorithm

```
Genetic algorithm(n,N,f,pm,pc)
  P = generate(n,N);
  while (!done())
    fv = evaluate(P,f,n,N);
    S = selection(P,fv,n,N);
    O = variation(S,n,N,pm,pc);
    P = replacement(O,P,n,N);
```

- Parameters
  - `n` The number of bits
  - `N` The population size
  - `f` The objective function
  - `pm` Probability of mutation
  - `pc` Probability of crossover

A Simple Genetic Algorithm

- Generate the initial population

```
generate(n,N)
  P = new population of size N;
  for i=1 to N
    P[i] = generate_random(n);
  return P;
```

- Evaluation

```
evaluate(P,f,n,N)
  fv = new array of N real numbers;
  for i=1 to N
    fv[i]=f(P[i],n);
  return fv;
```

Variation Operators

- Variation operators process selected promising solutions to generate new (hopefully more interesting) candidate solutions that share features with selected solutions
- Two basic principles
  - variation (introducing novelty)
  - inheritance (reusing the old)
- Variation in genetic algorithms consist of two operators
  - Crossover combines bits and pieces of promising solutions.
  - Mutation makes small perturbations to promising solutions.

Replacement (Elitism)

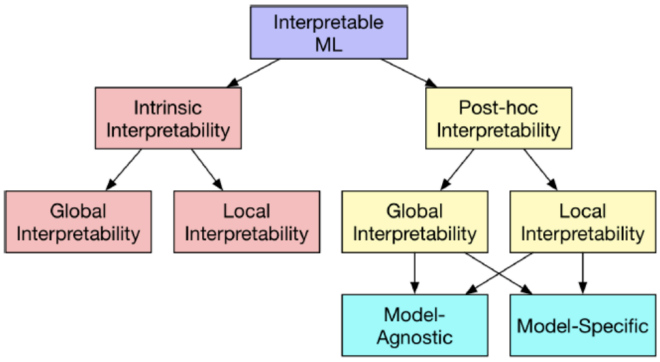
- Basic idea
  - Assume that the offspring population is smaller than the original population.
  - Replace worst strings in the original population by new offspring.
- Elitism
  - Elitist genetic algorithms preserve best solutions found so far.
  - This is one of elitist schemes.

Explainability

Why interpretability?

- Interpretability does often have a cost, so why do we need it?
  - Understanding the problem (causality) [Doshi-Velez and Kim, 2017]
  - Debugging and auditing (reliability and privacy)
  - Identifying bias (fairness)
  - Social acceptance and social interaction (trust)
- When don't we need explainability?
  - Well known problem/domain
  - Task with limited impact
  - Prevent cheating

Summary



Du, M., Liu, N., & Hu, X. (2019). Techniques for interpretable machine learning. Communications of the ACM, 63(1), 68-77.

Models - Pros & Cons

Linear Regression: pros and cons

- Pros
  - Model is very easy to understand (weighted sum)
  - Well studied
  - Optimal weights can be found analytically
- Cons
  - Interaction among features and non-linear relationships must be included by designing additional features
  - Might not work on complex dataset

Logistic regression: pros and cons

- Logistic regression can be extended to multi-class problems
- Pros
  - Model is rather easy to understand
  - Well studied and optimal weights can be found analytically
- Cons
  - Interaction among features and non-linear relationships must be included by designing additional features
  - Not sparse

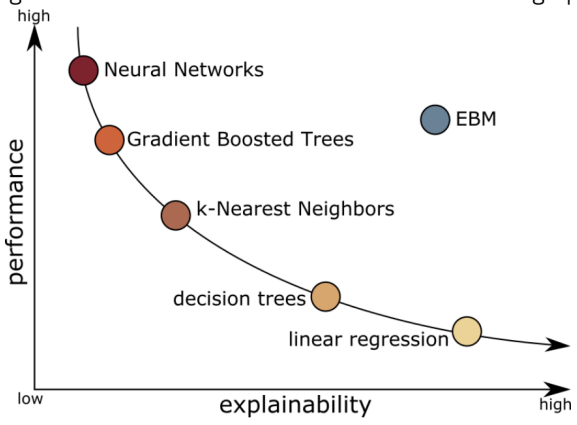
Decision Trees: pros and cons

- Pros
  - Allows to capture interactions between features
  - Human friendly and contrastive explanation
  - Can be very accurate if depth is large
- Cons
  - Does not capture well linear/smooth relationships
  - Often unstable with respect to changes in dataset
  - Interpretability highly dependent to depth

EBM

Explainable Boosting Machine (EBM)

- EBM combines boosting and Generalized additive model to achieve high performances with a glassbox model

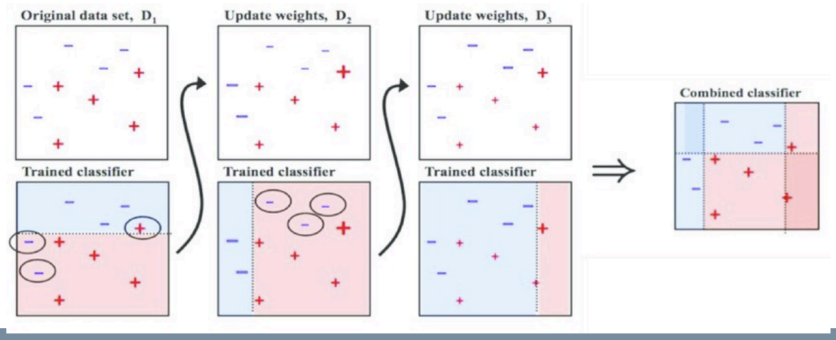


Explainable Boosting Machine (EBM): GAM and boosting

- Generalized Additive Models (GAMs)

$$g(E(Y)) = \beta_0 + f_1(x_1) + f_2(x_2) + ... + f_n(x_n)$$

- Boosting



Agnostic Methods

Permutation Feature Importance

Permutation Feature Importance

- Evaluates how important a single feature *j* is in a trained model as follows:

1. Train a model on a dataset
2. Shuffle the values of a single features in the dataset (i.e., column permutation)
3. Apply and evaluate the model both on original and on shuffled data.
4. Compute the feature importance of the feature as performance loss of the model on the shuffled dataset compared to the original dataset

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24

Permutation Feature Importance (2)

- How to perform permutation?
  - Half dataset swap
  - Generate N*(N-1) samples
- Should you evaluate the model on training set or test set?
- Pros
  - Global insight
  - Problem-independent
  - Considers interactions with other features
  - Does not require model retraining
- Cons
  - Linked only to model performance and requires labeled data
  - Is computed with possibly unrealistic data distribution
  - Correlated features share importance

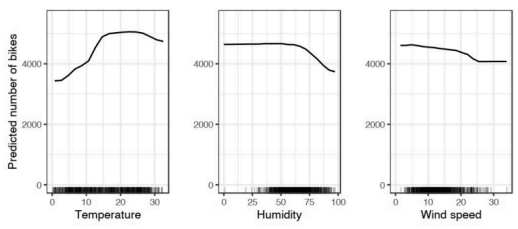
Partial Dependence Plot (PDP)

- While feature importance shows what variables most affect predictions, partial dependence plots show how a feature affects predictions.
- It works as follows:

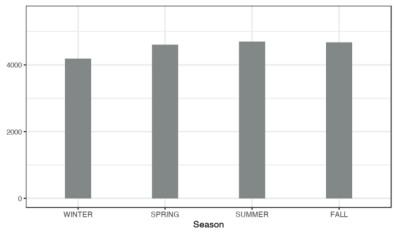
1. Train a model on data.
2. Run the model on each sample by changing only the value of target feature (choosing it from a set of values in the feature range)
3. Compute and plot the model output for each value of the feature.

- PDP can be computed also for two target features at once
- The variability of the PDP values (i.e., standard deviation or range) can be used as a measure for feature importance

PDP Examples



Numerical Feature



Categorical Feature

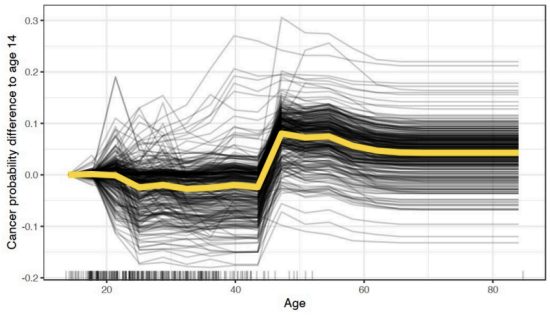
PDP: pros and cons

- Pros
  - Intuitive and easy interpretation
  - Easy implementation
  - Provides a casual relationship between feature and model prediction
- Cons
  - Shows the impact of 2 features at max
  - Might be not accurate in area where the feature distribution is small (thus is generally recommended to show also feature distribution)
  - Independence of the features is assumed
  - Shows the **average** effect on the prediction

Individual Conditional Expectation (ICE)

Individual Conditional Expectation (ICE)

- A major issue with PDP is that it only reflects the **average** behavior of the model
- ICE consists of multiple PDP plots, one for each sample in the dataset.
  - the values of all the features except the target one are fixed for each sample
  - the prediction values are plotted against the values of the target feature
- ICE can be plot together with PDP to highlights average behavior



- An **anchored** version of ICE can be plot using as a baseline the prediction on the left extreme of the feature range

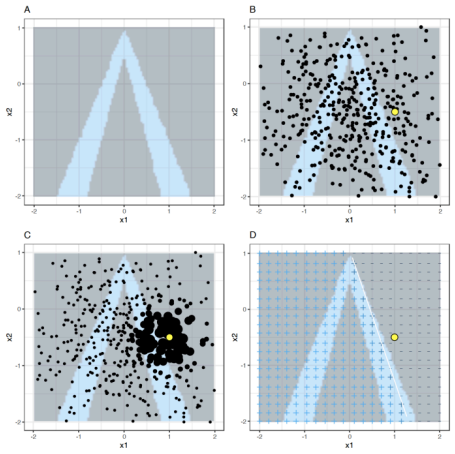
ICE: pro and cons

- Pros
  - Shows non-average/heterogeneous relationships
  - Very easy/intuitive to understand
- Cons
  - Works only for a single target feature
  - As for PDP, assumes features to be uncorrelated
  - Overcrowded plots could make it difficult to spot patterns

Local Surrogate (LIME)

- Surrogate models are trained to explain individual predictions:

1. Select your instance of interest for which you want to have an explanation of its black box prediction.
2. Perturb your dataset and get the black box predictions for these new points.
3. Weight the new samples according to their proximity to the instance of interest.
4. Train a weighted, interpretable model on the dataset with the variations.
5. Explain the prediction by interpreting the local model.



LIME: pros and cons

- Pros
  - Independent from explained model
  - Generated short explanation
  - Works for tabular data, text, and images
- Cons
  - How to define the neighborhood?
  - Data sampling does not account for correlation among features
  - Similar samples might result in different explanations
  - Complexity of explanation should be decided in advance

Shapley Values: introduction

- The underlying idea of SHAP values is being able to break down a model prediction by showing the contribution of each feature.
- In particular, the goal is computing for each feature j, the contribution ϕ_j such that:

$$\sum \phi_j = f(x) - E_x(f(x))$$

- In a linear model:

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$
$$\phi_i = \beta_i x_i - \beta_i \bar{x_i}$$

- How to compute it for any model?

Shapley Values: computation in practice

- To compute ϕ_j for an instance x:

Repeat for m = 1, ..., M

1. Draw random instance z from the dataset
2. Choose a random subset O of the features (not including feature j)
3. Create instance x_{+j} by replacing in x the features in O with values from z
4. Create instance x_{+j} by replacing in x all the in O and feature j with values from z
5. Update $\phi_j \; as \; \phi_j = \phi_j + 1/M \; (f(x_{+j}) - f(x_{-j}))$

Shapley Values: pros and cons

- Pros
  - Solid theoretical foundation
  - Contrastive explanations
  - Capture features interaction
- Cons
  - Not sparse (rely on all the features)
  - Computational expensive

SHAP (Shapley Additive Explanation)

- Combine the idea of LIME (learning local surrogate) with Shapley Values
- Several implementations are possible:
  - KernelSHAP is the general implementation that can be used with any model
  - TreeSHAP is faster implementation specific for tree-based model
  - DeepSHAP is similar to KernelSHAP but specifically devised for Deep Learning model