Final Study Guide

Wednesday, April 18, 2018 5:31 PM

Priority Study Topics:

- Clustering is most important
- Frequent Pattern Mining
- Text Mining
- SVD
- Anomaly Detection
- Rec Systems

Study 80% the first two, 20% of the last two.
From the first part of the course, have these key points down:

- How do you calculate similarity/proximity measures
- Application of classifiers to text mining and other topics

Final Review Session

- *Clustering:*
- The key idea with clustering is that it is an unsupervised learning problem; you do not know the ground truth like in classification where you have a training set with labels. Frequent Pattern Mining is also unsupervised.
- Cluster SSE measures the cohesiveness of your clusters, the tightness of the shape. Cluster SSB measures your separation of your clusters.
- SSE and SSB are used more for spherical clusters, where you have a concept of the "prototype" of the cluster, which is typically the mean of the points in that cluster. SSE + SSB = C. SSB captures how much structure you have captured in your model versus the SSE which is the error you have captured.
- The silloute coefficient measures validity as well.
- The foundational clustering algorithm is K Means, which works in some cases but fails in other cases.
- K Means is a specialization of EM Clustering, which is a type of mixture model clustering. K Means produces spherical clustering, meaning your covariance matrix is typically near 0 for off diagonal entries, with little covariance between attributes of points in the cluster.
- Hierarchical Clustering has two types:
    - Agglomerative

    - The Other One
- Agglomerative computes the proximity matrix of each point, lets each data point be a cluster, and keeps merging the two closest cluster and updating the proximity matrix. This keeps merging until a single cluster remains. To find the distance in the merge, you look at every cluster in the merged cluster, and either pick the data point with the smallest (min) or largest (max) distance to the point to be merged in.
- In a dendogram, you look at the lowest x parallel lines to find the lowest distance merges, then move up to higher x parallel lines to find higher merge distances.
- If you stop hierarchical clustering algorithm at a threshold of .15, the clusters with a distance less than .15 will be merged into one cluster, while those above will sit in their own clusters.
- 8.16 in the book is good practice:
- DBSCAN is a density based algorithm, where density is the number of points within a specified radius (epsilon). A point is a core point if it is near the center of that radius, border if it is near the border, and noise if it rides the border.
- The algorithm runs like follows:
    - Label all examples as either core, border, or noise points
    - Eliminate the noise points
    - Put an edge between all core points that are within an epsilon value of each other
    - Make each group of connected core points into a separate cluster
    - Assign each border point to one of the clusters of its associated core points
- DBSCAN has no concept of local factors. It operates off of global data. He skips everything past DBSCAN.
- Sparsification is when you break links between points in the data. Minimal spanning tree is a divisive clustering method that uses a graph based approach.
- Shared Nearest Neighbor measures distance in a different way, by measuring proximity by sharing a lot of points in common. This means you run KNN on every point, look at the neighbor intersection of all points and mark it as the similarity of two points if they are nearest neighbors of each other, and cluster based off of this.

- *Frequent Pattern Mining*
- Support and confidence were the main metrics for association analysis. We want to find all rules where the support is greater than the minimum support

(called frequent), and the confidence is greater than the minimum confidence (called high confidence).

- If an itemset is infrequent, than all of its supersets are infrequent by the Aprioiri principle. This means we can only generate frequent itemsets with frequent subsets that comprise its structure.
- A closed frequent itemset has the highest frequency/support if its supersets do not have a higher support than it. For closed frequency, you only need to look up one level.
- A maximally frequent itemset is an itemset that has no frequent supersets, meaning its supersets have a support lower than the minimum support.
- Maximal frequency implies closed frequency. Maximally frequent itemsets are a subset of the closed itemsets.
- Given the maximal frequent itemsets, you can derive all of your frequent itemsets. This is why we care about them. We care about closed frequent itemsets because they allow us to determine which itemsets are frequent and what their support its.
- To calculate the confidence of a rule, create a fraction. The denominator is the number of transactions that have the consequent AND the antecedent in them. The numerator is the total number of rules with the antecedent in them.
- He skipped a lot after frequent pattern mining to sequential pattern mining.
- The key thing to know is that the algorithm has 4 steps where you start with sequences with length 1, then move to sequences of length 2, 3, 4…
- You then basically run the frequent pattern mining on them

- *SVD*
- Know what you can use a SVD for and the basic concepts behind it.
- The columns represent abstract, latent concepts within the dataset. In data, there are underlying concepts that help define things in your data. You are trying to uncover hidden structure in your data. U represents the concepts for the rows, V represents the concepts for the columns, and Sigma is the matrix in the middle. You wont be asked to calculate anything substantial. Same thing for anomaly detection, recommendation systems.

Proximity Measures

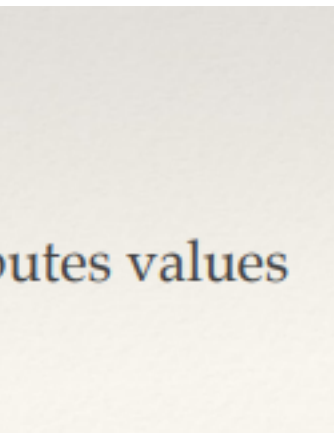
- **Proximity** refers to either similarity or dissimilarity between points/attribute values. Distance is a common term for dissimilarity. To scale dissimilarities, use this formula:
    - (original dissim - minimum dissim value) / (max dissim value - min dissim value), where the max and mins are the values from your original range of dissimilarities

dissimilarities.

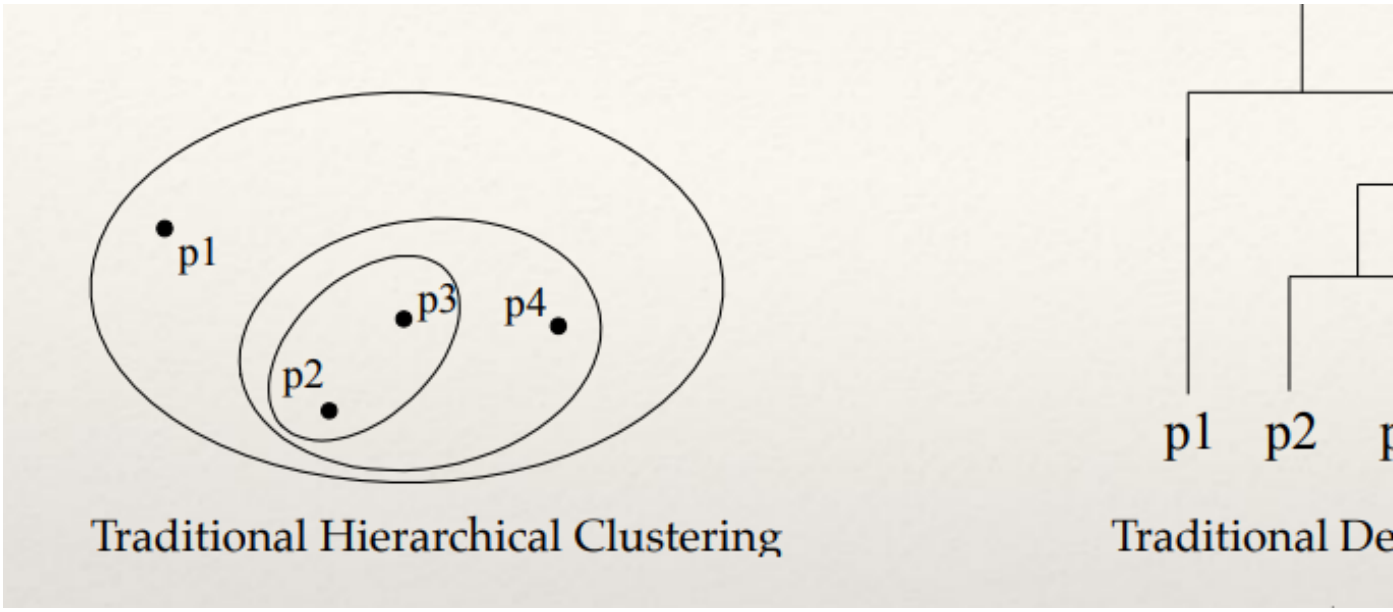
- The Euclidian distance is:
  - $$dist = \left(\sum_{k=1}^{n} | p_k - q_k |^r \right)^{\frac{1}{r}}$$
  - Where r = 2. You iterate through the attribute values, adding them together after subtracting them, squaring them. Then, you square root the result.
- The simple matching coefficient is used only for binary attributes. It is denoted as the number of true positives and true negatives divided by the number of all outcomes. Jaccard is an offshoot, and only calculates the number of true positive matches, divided by the number of all attributes except for true negative matches. It is more appropriate for attribute vectors with a lot of matching true negatives:
- $$SMC = \text{number of matches} / \text{number of attributes}$$
  $$= (M11 + M00) / (M01 + M10 + M11 + M00)$$
  $$J = \text{number of 11 matches} / \text{number of not-both-zero attrib}$$
  $$= (M11) / (M01 + M10 + M11)$$
- Cosine similarity treats the two attribute rows in the dataframe as vectors. It is the dot product of the two vectors divided by the product of their norms:
- $$\cos(d_1, d_2) = (d_1 \bullet d_2) / ||d_1|| \, ||d_2||$$
- Remember that with proximity, different attributes have different scales so you should scale your data before computing distance.

Clustering

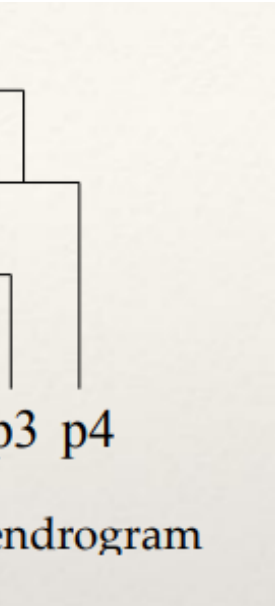
- The basics of clustering is finding groups of objects such that the objects in a group will be similar to one another and different from the objects in other groups. We can use clustering for understanding, like grouping related documents. We can also use it for utility, like abstracting individual data objects to those clusters in which the objects reside.
- Partitional clustering divides data objects into non overlapping subsets such that each data object is in exactly one subset. Hierarchical clustering creates a set of


utes values

nested clusters organized as a hierarchical tree.

- 

- The **dendogram** works as follows: The first cluster made is p2 and p3, so they have the lowest line on the graph. Then, p4 is added to the next cluster with p3 and p2 in it, meaning it merges with the smaller cluster. Then p1 is finally added, which has the highest line.
- **K Means** clustering is a partitional cluster algorithm. It works as follows:
    - Each cluster has a centroid. Select K points as the initial centroid, which are just random points in your dataset.
    - Assign all points in your dataset to the nearest centroid.
    - Update the centroids to be the new means of every cluster you just made.
    - Repeat until the centroids change by a very small cutoff amount.
- K Means is bad at detecting non-globular shapes, and has trouble with outlier data.
- Mixture models are another way to do clustering. They assume that the underlying data is generated as a result of some statistical process. MM want to find a stat model that best fits the data, where the model is described in terms of a distribution and a set of parameters for the distribution.
- MM are a particular type of stat model for this. It views the data as a set of observations from a mixture of different probability distributions, where each distribution corresponds to a cluster, the parameters of the model describe cluster characteristics, and the most common form is the Gaussian mixture, where each cluster is described by a mean vector and a covariance matrix.
- The **Expectation Maximization** algorithm can use the maximum likelihood to estimate the parameters for a mixture model. EM calculates the probability that each point belongs to each distribution and then uses these probabilities to



endrogram

compute a new estimate for the parameters. This continues until the parameters converge.

- The algorithm runs as follows:
    - Select an initial set of model parameters
    - Until the parameters converge:
        - EXPECTATION STEP: For each object, calculate the probability that the object belongs to each distribution
        - Maximization Step: Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
- Remember that EXPECTATION is like cluster assignment, and Maximization is like recomputation of cluster centroids in Kmeans.
- There are two types of **hierarchical clustering:**
    - Agglomerative, which starts with the points as individual clusters, and at each step, merges the closest pair of clusters until only one cluster is left
    - Divisive, which starts with one, all inclusive cluster, and at each step, splits a cluster until each cluster contains a point.
- Traditional hier algorithms use a similarity or distance matrix, which has rows and columns where the rows are one point, and the columns are another point, and the cells are the distances between those two points, and the diagonal is 0 as those points are the same point.
- **Agglomerative** is the most popular:
    - Compute the proximity matrix
    - Let each data point be a cluster
    - Repeat:
        - Merge the two closest clusters
        - Update the proximity matrix (the 2 merged points are now a set, as they are a merged cluster)
    - Until only one cluster remains.
- We can define **inter-cluster similarity** in a couple of ways:
    - MIN, or single link, where you choose the highest similarity or the shortest distance from any point in a merged cluster to the other point. This means that the point in the merge with the smallest distance assigns its distance as the merge cluster distance.
    - MAX, or complete link, uses the largest distance and acts like MIN in every other way
    - Group Average finds the distance between every point in a merged cluster and every other point not in the cluster, and uses the average as the new

distance
    - Distance between centroids
- **DBSCAN** is a density based algorithm. The density is the number of points within a specified radius called epsilon.
- There are three types of points:
    - Core points, which are points with more than a specified number of points (MinPts) within epsilon.
    - Border points, which has fewer MinPts within epsilon, but is in the neighborhood of a core point.
    - Noise points, which are any point that is not the above two.
- The algorithm runs as follows:
    - Label all examples as either core, border, or noise
    - Eliminate all noise
    - Put an edge between all core points within epsilon of each other
    - Make each group of connected core points into a separate cluster
    - Assign each border point to one of the clusters of its associated core points
- DBSCAN does not work well for clusters of varying densities and high dimensional data.
- **Sparsification** is when you break links between points in the data. Minimal spanning tree is a divisive clustering method that uses a graph based approach. This is done by making the proximity matrix a graph, with each column being a node, and each distance between each column and row being an edge. Sparsification breaks all links that are low similarity enough to form different clusters which themselves are represented by subgraphs.
- You can construct a **minimum spanning tree** for the graph generated by the dissimilarity matrix (high value means low proximity). To generate clusters, break links in the MST corresponding to the largest dissimilarity. Do this until you have enough good clusters.
- **Shared Nearest Neighbor** measures distance in a different way, by measuring proximity by sharing a lot of points in common. This means you run KNN on every point, and if two points are among the nearest neighbors of each other, their similarity is the number of neighbors they share. If they are not among the nearest neighbors, their similarity is 0.
- Cluster cohesion is how closely related the objects in a cluster are, and is measured with the Sum of Squared Error for a cluster. Total SSE is the sum of all of these cluster SSEs: (ci is the centroid)

- $$\text{Cluster SSE} = \sum_{\mathbf{x} \in C_i} dist(\mathbf{c}_i, \mathbf{x})^2$$
- Cluster Separation measures how distinct or well separated a cluster is from other clusters, which is measured by Cluster SSB:
- $$\text{Total SSB} = \sum_{i=1}^{K} m_i \, dist(\mathbf{c}_i, \mathbf{c})^2$$

where m_i is the size of cluster i

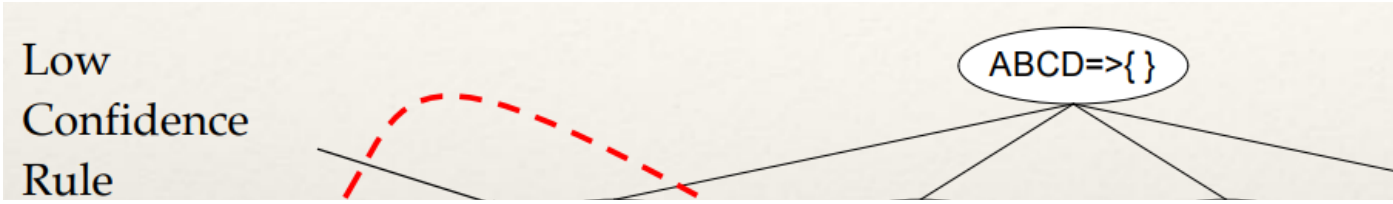
Frequent Pattern Mining

- The goal of FPM is to, given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction. So, given these items, what is the likelihood that we'll have a new item.
- An itemset is a collection of one or more items.
- Support count is the frequency of an occurrence of an itemset. So the support count of 1 in 1131 is 3.
- Support is the fraction of transactions that contain an itemset. So its just the support count divided by the number of itemsets.
- A frequent itemset is an itemset whose support is greater than or equal to a threshold called minimum support.
- An association rule is an implication expression of the form X -> Y, where X and Y are the itemsets. You can evaluate a rule by its **support**, which is the number of itemsets that contain both X and Y, and its **confidence**, which measures how often items in Y appear in transactions that contain X.
- Mining association rules is a two step approach: First you have to generate frequent itemsets, who have a support greater than the minimum support threshold. Then, you have to do rule generation, where you generate high confidence rules from each frequent itemset.
- The Apriori principle states that if an itemset is frequent, then all of its subsets must also be frequent. That means if we find the smallest candidate itemset

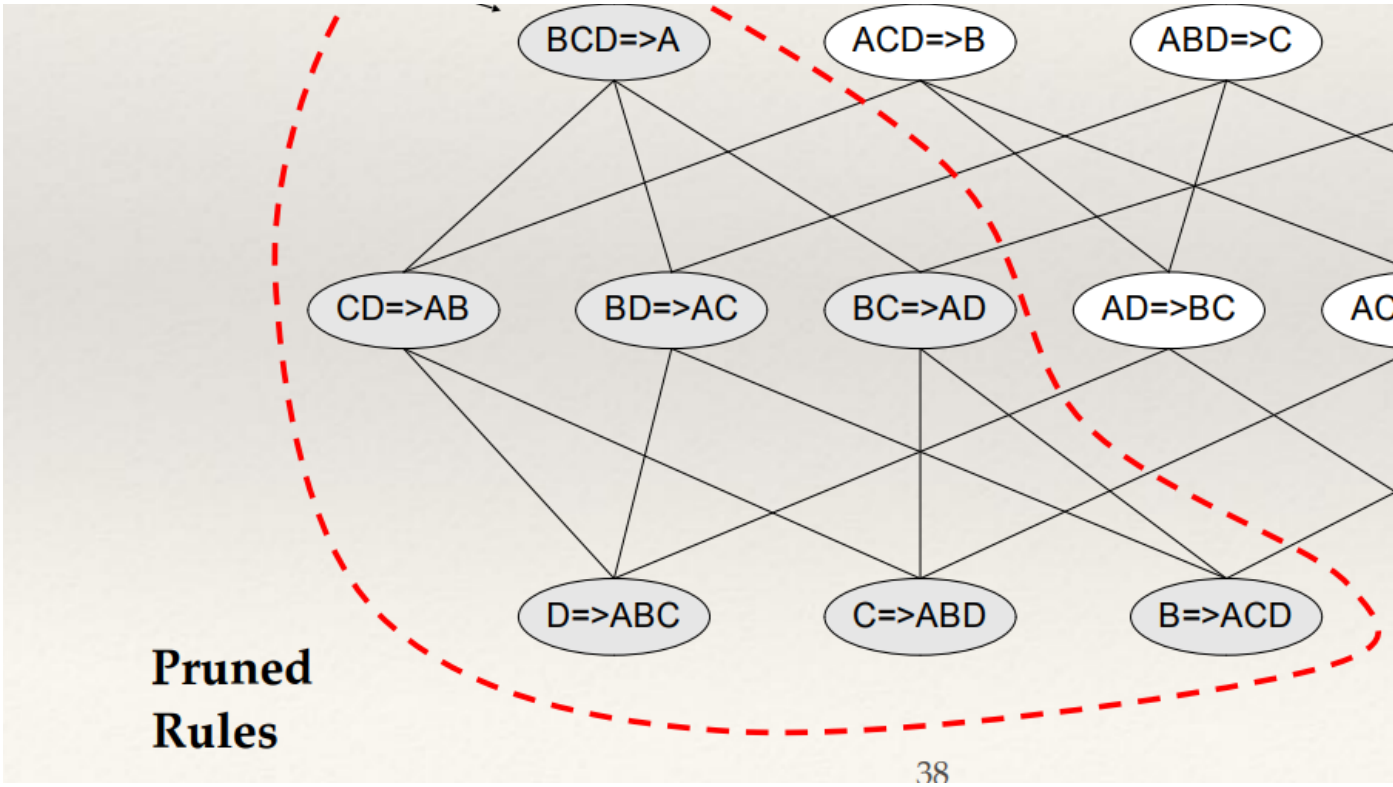
that is infrequent, all of its superset candidates are also infrequent. This is due to the anti-monotone property of support.

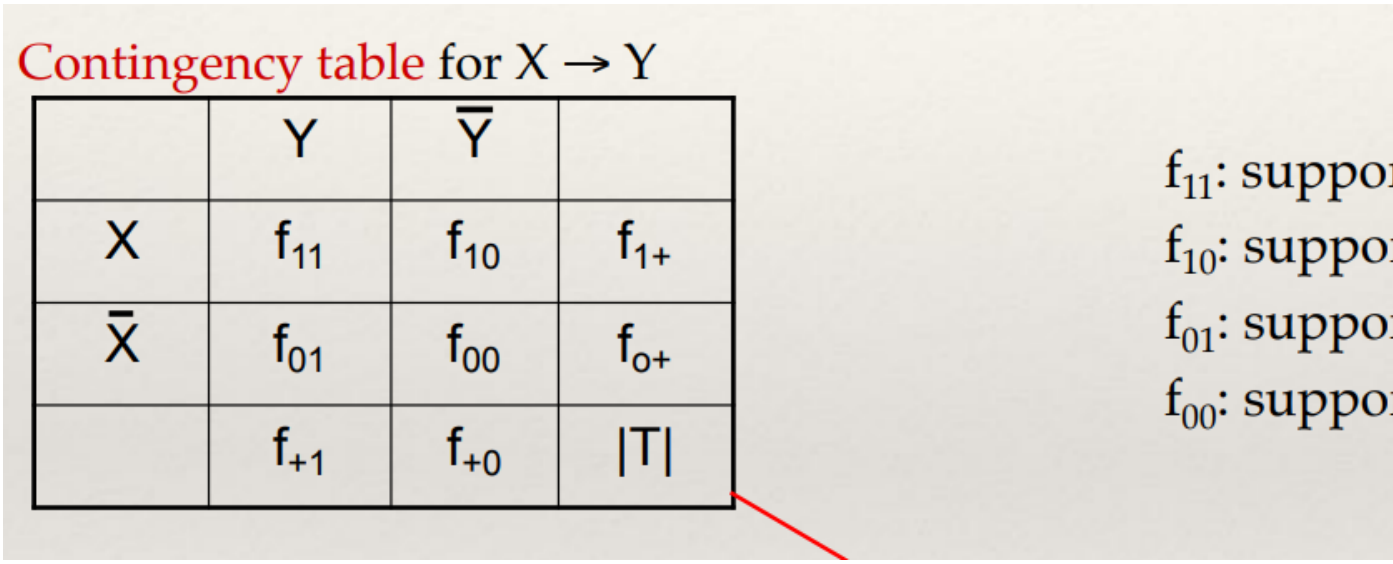
- The Apriori method uses this principle to generate all frequent itemsets:
    - Let k = 1, and generate frequent itemsets of length 1. Then, until no new frequent itemsets are identified:
        - Generate k+1 length candidate itemsets from the length k itemsets
        - Prune the candidates that have subsets of length k that are infrequent
        - Count the support of each dataset by scanning the DB
        - Eliminate the candidates that are infrequent whose subsets were not infrequent.
- An itemset is maximal frequent if none of its immediate supersets are frequent. An itemset is closed if none of its immediate supersets have the same support as it. Maximal frequent itemsets provide a compact representation of the frequent itemsets, but do not contain info about the support of their subsets. Closed frequent itemsets provide a minimal representation of frequent itemsets without losing support information.
- Rule generation generates all rules that have a confidence higher than the minimum confidence parameter. Each rule generated is a frequent itemset.
- RG takes all possible candidate rules that can be generated by a frequent itemset, and checks their confidence against the DB. The amount of total rules that can be generated are 2^(length of itemset) - 2.
- Rule confidence of items generated from the same itemset has an anti monotone property. The confidence of a rule with more items on the left of the arrow will always be higher than one with less.
- ❖ e.g., $L = \{A,B,C,D\}$:

  $c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c($
- So the Apriori method for rule generation is to take two rules and merge them by the number of items in the consequent. Candidate rules are generated using two rules with k items in the consequent to make a new rule with k + 1 items in the consequent. If the new rule has a high confidence, we keep going. But if a k rule has a low confidence, we do not have to go further with rule generation:

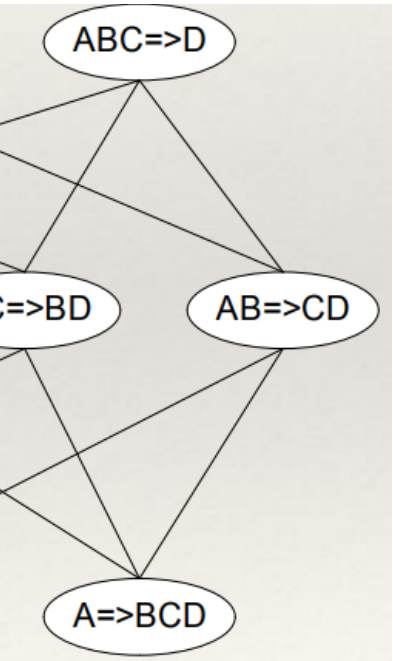


(A → BCD)

- 

- A **contingency table** can be used to compute rule interestingness. It is made like so:
- 

- Confidence can be misleading. Another measure exists, called lift. Lift is equal to:
- Lift(X - Y) = P(Y|X)/P(Y), where P(Y|X) is f11 / f1+.
- Sequences can be mined as well. A sequence is an ordered list of elements/transactions. Each element is a collection of events/items, with each element attributed to a specific time or location.
- A sequence (1) is contained in another sequence (2) if each element in 1 is a subset of its nth corresponding element in 2. Also note that the correspondence does not have to be exact, in the below example in row 1, the subsequence



rt of X and Y
rt of X and $\overline{Y}$
rt of $\overline{X}$ and Y
rt of $\overline{X}$ and $\overline{Y}$

does not have to be exact, in the below example in row 1, the subsequence <{2}, {8}> would work. All that has to be maintained is the ordering:

Data sequence	Subsequence	Co
< {2,4} {3,5,6} {8} >	< {2} {3,5} >	
< {1,2} {3,4} >	< {1} {2} >	
< {2,4} {2,4} {2,5} >	< {2} {4} >	

- The support of a subsequence w is defined as the fraction of data sequences that contain w. A sequential pattern is a frequent subsequence.
- Sequential pattern mining is the following problem: You have a DB of sequences, and a user defined minimum support threshold. You want to find all subsequences with support greater than that minimum support.
- The amount of k sized subsequences that can be extracted from an n size sequence is equal to n choose k.
- Generalized Sequential Pattern Mining runs like so:
    - Make the first pass over the sequence database D to yield all 1-element frequent sequences.
    - Repeat the following until no new frequent sequences are found:
        - Merge pairs of frequent subsequences found in the k-1th pass to generate candidate sequences that contain k items
        - Prune candidate k sequences that contain infrequent k-1 subsequences
        - Make a new pass over the sequence database D to find the support for these candidate sequences
        - Eliminate candidate k sequences whose actual support is less than the minimum support threshold

Text Mining

- Text mining refers to the process of generating interesting and useful insights from text data. Text mining usually involves the following steps:
    - Structuring the input text (usually parsing, along with the addition of some derived linguistic features)
    - Deriving patterns from that data
    - Interpreting the output
- Uses for text tasks:

ontain?
Yes
No
Yes

- Uses for text tasks:
    - Text categorization/classification: assign a document to one or more classes
    - Text clustering: find clusters of similar documents
    - Concept/entity extraction of artifacts from text
    - Sentiment analysis
    - Keyword based association analysis
    - Similarity detection
    - Link analysis
    - Sequence analysis
    - Anomaly detection
    - Hypertext analysis
- Bag of words takes raw text data, and converts it to a key value map where the key is the word and the value is the number of occurrences of that word in the raw data. This loses all order-specific information, all you're looking at are keywords, and limits the context of the original data.
- This approach is still useful.
- NLP methods also attempt to use computers to derive meaning from text data in a good way. The bag of words approach is shallow, very limited amounts of information are retained from the original text beyond keywords. NLP methods take into consideration a deeper level of context about the language.
- Our vector space model is represented by:
    - A term vector, where a term is a word or phrase. Each term is one dimension (column), with n terms making a n dimensional space. The element of a term vector corresponds to the term weight/importance. Preprocessing the data is often very helpful, where you remove stop words, and do word stemming, where you convert something like computer or computing to the word compute.
- There's a couple of heuristics that we can use to weight terms in the vector:
    - Term frequency, where if a term is more frequently featured in a document, it is more important. This just means counting the words in the document.
    - Inverse Document Frequency, where if a term is less frequent among documents, meaning a term over every document in your parse set, then it is more descriptive and more important. This means that a smaller word column total is important.
    - The Raw term frequency is how many times term t appears in doc d. Normalization is the raw frequency divided by the sum of all the frequency of every word in the document. This means your cell value is

(cell value)/(sum of all cells in row). Double normalization multiplies the cell value by 0.5, divides it by the largest cell in the row, and then adds 0.5 to it.

- To calculate the IDF, we do log(total number of documents / docs with term t appearing), with the base of the log being base 10.
- TF-IDF weighting just takes the IDF, and multiplies it by the value in the term cell.
- To compare the similarities of two documents, we can take either the dot product or the normalized dot product of the document rows (feature vectors) after they have been transformed using TF-IDF weighting:
- **1. dot product** $Sim(D_i, D_j) = \sum_{t=1}^{N} w_{it}$
- **2. normalized dot product (or cosine)** $Sim(D_i, D_j) = \frac{\sum}{\sqrt{\sum_{t=1}^{N}(}}$
-
- Data cleansing work is done a lot in text mining:
  - Remove stop words, like the and be and and.
  - Perform word stemming, where you change words to a base word. These words are different forms of the base word, like misspellings and other types
- The probability that a document is class C is P(D|C)*P(C). P(C) is equal to the number of documents in the training examples of class C, divided by the number of documents in the training examples. P(D|C) is given by the two excerpts below:
- A probabilistic model assumes that documents are labeled with predefined categories. The category hypothesis space has all of the categories possible. We want to calculate the most likely category for a single document by using the Multi Bernoulli model, where the probability that a document model exists for class Ci:

$$\times w_{jt}$$

$$\frac{\sum_{t=1}^{N} w_{it} \times w_{jt}}{\sqrt{(w_{it})^2} \sqrt{\sum_{t=1}^{N} (w_{jt})^2}}$$

$$|V|$$

- $$p(D=(x_1,\ldots,x_{|V|})\mid C)=\prod_{i=1}^{|V|} p(w_i=x_i\mid C)=\prod_{i=1,x_i=1}^{|V|} p(w_i$$

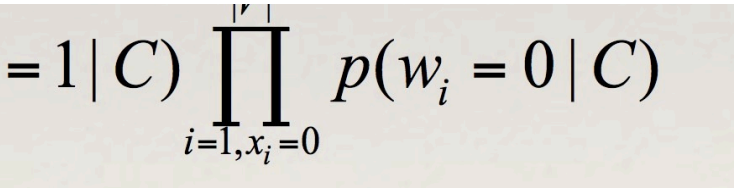
- To determine the individual probabilities of each term existing in a category of document is (these are the probabilities in the cumulative products in the multi Bernoulli model):

- $$p(w_j=1\mid C_i)=\frac{\sum_{d\in E(C_i)}\delta(w_j,d)+0.5}{|E(C_i)|+1} \qquad \delta(w_j,d)=\begin{cases}1 & if\ w_j\ occurs\ in\ d\\ 0 & otherwise\end{cases}$$

- The probability of a class being a specific category given the document model is:
    - P(DocumentModel | Class) * probability of category
    - The first prob in this is the Multi Bernoulli model.

Singular Value Decomposition

- The goal of dimensionality reduction is to discover the axis of the data.
- Singular Value Decomposition takes in (r is the rank of matrix A, which is the number of linearly independent columns):
    - A[mxn], which is the input data matrix. Think m documents, n terms or something like this.
    - U[mxr], which are the left singular vectors. M documents, r concepts.
    - Sigma[rxr] which are singular values, in an rxr diagonal matrix, where each diagonal is the strength of its respective concept.
    - (V[nxr])^T, which are the right singular vectors, n terms, r concepts.
- R is the rank of the mxn matrix A. Slide 6 shows the LinAlg to compute the SVD of a matrix. Next we do an example.
- Users to Movies is our example. We have rows being the users, and the columns being their rankings on movies.
- To get the SVD in R for a matrix or dataframe, just use SVD(matrix/dfname)
- The matrix U's column describe a different concept each for every row. In the example, the first concept is Sci Fi movies, and the second is Romance movies. The last column maps to the interactions between concepts for each user.
- The matrix A gives the strength of each concept in the diagonal. In the example,



scifi is the strongest, the interaction is the weakest.

- The matrix V is the movie to column concept. Or, in general, column label to concept label similarity matrix. The rows are for each concept. The columns are for each column label. The last row is the interaction.
- The idea behind SVD is to use the rows of V^T to create new axis to project the rows of A onto them.
- (U * Sigma)^T gives the coordinates of the first right singular vector of the points in the projection axis. SVD removes the smallest value in Sigma and its corresponding column in U and its corresponding row in V^T. You would take these, multiply them together again, and you get a best approximation of A from the product.
- The Frobenius Norm can measure what the best approximation is after running SVD.
- A theorem exists that states that the rank of an SVD reduced matrix is the best approximation of the original matrix.

Anomaly Detection

- Anomalies/outliers are the set of data points that are considerably different than the remainder of the data.
- Using the class label (normal/anomaly):
    - You can have supervised anomaly detection, with a training dataset with example anomalies
    - Unsupervised anomaly detection, where no class labels are available, and the goal is to assign a score to each example that reflects the degree to which that instance is anomalous
    - Semi supervised anomaly detection, where some data, but not all is labeled, for example starting out with a small set of representative normal objects.
- Some of the challenges include:
    - Number of outliers in the data
    - Unsupervised situations are mostly encountered, so validation can be really hard like clustering
    - It's like finding a needle in a haystack.
- An object can globally look normal but in a local perspective, it can be an anomaly, and vice versa.
- An example anomaly might be anomalous for some attributes but not for others. You can measure the degree of anomalimity.
- **Masking** is when the presence of several anomalies masks the presence of all of

them. **Swamping** is when normal objects are mistakenly categorized as anomalies.

- Graphical based approaches to anomaly detection are good to detect anomalies. Finding outliers on box plots and scatterplots.
- Statistical approaches involve anomalies as objects in a distribution that have low probability in the statistical model (like the normal curve) that describe the data. The issue with this approach is the ambiguity of the outliers: Should it be points outside the 95% confidence interval? 99%?
- The Likelihood Approach does the following:
    - Assumes your dataset D has samples from a mixture of two probability distributions:
        - The majority distribution
        - The normal anomalous distribution
    - Assume all the data points belong to the majority
    - Let LLt(D) be the log likelihood of D at time t
    - For every point in majority distribution, move it to the normal anomalous distribution
    - During that move, detect the change in the log likelihood, and if the difference is more than some preset threshold, then that point is declared as an actual anomaly, and moved permanently to the normal anomalous distribution.
- In many cases, the distribution of the data may not be known for this approach. For high dimensional data, it may not scale well.
- A proximity based anomaly says that an object is an anomaly if it is far away from most points in the dataset. One approach to this is nearest neighbors distance. To classify anomalies, either set the point for which there are fewer than p neighboring points within a distance D as an anomaly, or look at the average distance between the point and its k nearest neighbors or kth nearest neighbor for every point and then label the highest distances as anomalies.
- The density based approach sets the anomaly score is the inverse of the density around an object. We define density as the reciprocal of the average distance to the k nearest neighbors. If this distance is small, the density is high and vice versa. Density can also used a fixed value, that helps determine the density when it is defined as the number of objects around that object that are within a fixed distance of an object.
- Neither of the above methods, but cannot identify anomalies correctly when regions of differing densities exist in a dataset.
- The local outlier factor is a way to get around this. It is computed as the ratio of the average density of the nearest neighbors of point p and the density of p

the average density of the nearest neighbors of point p and the density of p itself.
- Cluster based anomaly detection has multiple approaches:
    - Discard small clusters that are far from other clusters
    - Discard clusters smaller than a minimum size
    - Cluster all objects and then assess the degree to which an object belongs to any cluster. Basically, this states that an object is a cluster based anomaly if the object does not strongly belong to any cluster.

Recommendation Systems

- Content based systems examine properties of the items they recommended. Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended are those preferred by a similar user.
- There are many types of recommendations:
    - Editorial and hand curated
    - Simple aggregations like top 10 lists
    - Tailored to individual users like Amazon and Netflix suggestions
- The formal model for a recommendation is that X is a set of customers, S is a set of items, and a utility function exists such that u: X x S -> R where R is a set of ratings that is totally ordered, like 0-100 percent or 0-5 stars.
- The utility matrix contains 0-1 real number values in each cell, and matches a user up to a product. Users are rows, products are columns.
- Key challenges in recommendation:
    - Gathering known ratings from the utility matrix
    - Extrapolating unknown ratings from the known ones in the UM. We want to find high unknown ratings so we can sell people shit.
    - Evaluating extrapolation methods. How do we know what we recommend is the good shit that people want?
- Gathering ratings can be either explicit or implicit. We could ask people to rate items (which never works), or we can learn ratings from user actions.
- The UM is sparse. Not every user has rated every item the site can sell. New items have no ratings, and new users have no history with which we can craft a recommendation with.
- Content based recommendations suggest items to customer x similar to previous items rated highly by x. Think movie recommendations, linking to "similar content". To do this, create an item profile for every item. The profile is a vector of features. You have to pick important features, like for movies, it would be actors, directors, genre and so forth.

- User profiles are vectors that have the same components as the item profiles which describe the user's preferences. These are typically the weighted average of rated item profiles:

User Purchases

User	Sci-Fi	Romance	Comedy
A	1		
A	1		
A		1	
A		1	
A		1	
B			1
B			1
B		1	

User P

User	Sci-Fi
A	0.4
B	

-

- If the user purchases are not just 1 indicators of having purchased an item with a specific category, but a rating, you can normalize the cell value by subtracting the overall average item rating for that specific user.
- Content based is good as you don't need data on other users to run it. You can recommend to unique users with weird tastes, and you can find new and unpopular items. However, building new user profiles is hard, recommending things outside a content profile is even harder, and choosing features of items is tough.
- Collaborative filtering considers user x. It finds a set N of other users whose ratings are similar to the ratings of x. It estimates ratings for X based on the ratings of users in N.
- To find similar users, both Jaccard and cosine similarity measures fail. The only good measure is the Pearson correlation coefficient:

- **Pearson correlation coefficient**
  - S_{xy} = items rated by both users x and y

rofiles

Romance	Comedy
0.6	
0.33	0.67

-

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

- To predict user x's rating for item i based on the set N of k users who are most similar to user x:

$$r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$$

-

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Shorthand

$s_{xy} = sim$

- Another way to do collaborative filtering is item to item. For item I, find other similar items. Estimate the rating for item I based on ratings for similar items using methods alike those for user to user collaborative filtering.
- Item to item works better than user to user, as users have multiple tastes. Collaborative filtering is good because it works on any kind of item. It is bad because it uses a cold start, with a need for enough users in the system to find a match. The matrix it uses is sparse, and cannot rate items that have not been previously rated. Also, it recommends items based on popularity, not to individual taste.

$\bar{r}_x$, $\bar{r}_y$ … avg. rating of **x**, **y**

d:

$n(x, y)$