CMSC 724: Data Mining

Amol Deshpande

University of Maryland, College Park

March 24, 2009
Outline

1. Overview
2. Association Rules
3. Clustering
4. Classification
   - Naive Bayes Classifier
5. Data Mining + Query Evaluation
Data Mining

- Going beyond the “query-response” model
  - “Tell me what is interesting in this huge dataset”?
  - Also called “knowledge discovery”
- Many applications
  - Marketing, fraud detection etc.
- Much work in the Machine Learning and Statistics literature
- Three of the most common problems addressed in the area:
  - Clustering
  - Association Rules
  - Classification
Outline

1. Overview

2. Association Rules

3. Clustering

4. Classification
   - Naive Bayes Classifier

5. Data Mining + Query Evaluation
Consider “market basket” data
- Each “transaction” contains the list of “items” bought by a customer in one visit
- Not normalized (transactions contain diff no. of items)
- Example:
  - (customer1, “2/2/2 5pm”, “milk”, “bread”, ...)
  - (customer2, “2/3/2 5:10pm”, “milk”, “coffee”, ...)
Consider “market basket” data

- Each “transaction” contains the list of “items” bought by a customer in one visit
- Not normalized (transactions contain diff no. of items)
- Example:
  - (customer1, “2/2/2 5pm”, “milk”, “bread”, …)
  - (customer2, “2/3/2 5:10pm”, “milk”, “coffee”, …)

Association Rule: $X \implies Y$, with conf $c\%$, support $s\%$

- Confidence: $c\%$ of all tuples that contain $X$ contain $Y$
- Support: $s\%$ of all tuples in the table contain $X \cup Y$
- $X$ and $Y$ are sets of items (“itemsets”), with $X \cap Y = \emptyset$
Association Rule Mining

Goal:

Find all association rules with at least a given confidence (say 90%) and a given support (say 5%)
Goal:
- Find all association rules with at least a given confidence (say 90%) and a given support (say 5%)

Fast Algorithms for Mining Association Rules; Rakesh Agrawal and Ramakrishnan Srikant; VLDB 2004
- Presented an algorithm called “Apriori”
  - For finding the itemsets with the specified support
- Probably the first “scalable” data mining paper
- One of the highest cited papers in all of CS
- Came out of Database literature, not Machine Learning
Basic Apriori Algorithm

- Say the required support is 5%
- Observation: A set $X$ cannot have sufficient support if $Y \subset X$ doesn’t have sufficient support
Basic Apriori Algorithm

Say the required support is 5%

Observation: A set $X$ cannot have sufficient support if $Y \subset X$ doesn’t have sufficient support

Algorithm:

1. Start by counting all items that occur in 5% transactions (say $\{l_1\}$, $\{l_2\}$, $\{l_3\}$)
2. Generate a set of candidates for the next step by combining the above itemsets ($\{l_1, l_2\}$, $\{l_1, l_3\}$, $\{l_2, l_3\}$)
3. Make another pass and check if the candidate itemsets appear in 5% transactions (say $\{l_1, l_2\}$, $\{l_1, l_3\}$ appear)
4. Repeat steps 2 and 3. (In the example, there are no more candidate sets. $\{l_1, l_2, l_3\}$ cannot occur in 5% transactions, because $\{l_2, l_3\}$ doesn’t).
Association Rule Mining

- Basic Apriori Algorithm:
  - Step 2 done using SQL queries and self-joins
    - Essentially a self-join between the itemsets found at previous step
    - Number of joins limited (and size of the relations decreases)
  - More efficient algorithms presented in the paper

- Much more work on this topic since that paper
  - 5197 citations on Google Scholar
  - Compare with: Garey and Johnson (18226) or Introduction to algorithms (13786)
Basic Apriori Algorithm:
- Step 2 done using SQL queries and self-joins
  - Essentially a self-join between the itemsets found at previous step
  - Number of joins limited (and size of the relations decreases)
- More efficient algorithms presented in the paper

Much more work on this topic since that paper
- 5197 citations on Google Scholar
  - Compare with: Garey and Johnson (18226) or Introduction to algorithms (13786)
Outline

1. Overview

2. Association Rules

3. Clustering

4. Classification
   - Naive Bayes Classifier

5. Data Mining + Query Evaluation
Clustering

- Given a set of tuples and $k$, group them into $k$ “clusters”
  - So that some distance metric is minimized
- Unsupervised: You are only given $k$
- Cluster IDs have no intrinsic meaning (though one can be provided later)
- Commonly used distance metrics:
  - For numerical data: Euclidean distance, Manhattan distance, $L_{\infty}$ distance
  - For categorical data: Map to reals

Number of clusters?

The algorithms find exactly $k$ clusters
Too many clusters not very useful
Trying to have few clusters results in large diameters
Clustering

- Given a set of tuples and \( k \), group them into \( k \) “clusters”
  - So that some distance metric is minimized
- Unsupervised: You are only given \( k \)
- Cluster IDs have no intrinsic meaning (though one can be provided later)
- Commonly used distance metrics:
  - For numerical data: *Euclidean* distance, *Manhattan* distance, \( L_\infty \) distance
  - For categorical data: Map to reals

Number of clusters?

- The algorithms find exactly \( k \) clusters
- Too many clusters not very useful
- Trying to have few clusters results in large diameters
Clustering

- k-means vs k-center vs k-median
  - k-means: The center of a cluster is the “mean” of the points assigned to it
    - May not be a real point
Clustering

- k-means vs k-center vs k-median
  - k-means: The center of a cluster is the “mean” of the points assigned to it
    - May not be a real point
  - k-center and k-median:
    - The cluster centers must be chosen from the points to be clustered
    - k-center: minimizes the largest distance between a point and its cluster head
    - k-median: minimizes the sum of distances between points and their cluster heads
Clustering

- k-means vs k-center vs k-median
  - **k-means**: The center of a cluster is the “mean” of the points assigned to it
    - May not be a real point
  - **k-center and k-median**: The cluster centers must be chosen from the points to be clustered
    - **k-center**: minimizes the largest distance between a point and its cluster head
    - **k-median**: minimizes the sum of distances between points and their cluster heads

- The latter two considered harder

- *Facility Location Problem*: The number of centers is not fixed, but there is a cost to choose a new center
Clustering

- “k-means” algorithm
  - At any point, have $k$ “centroids”, one for each cluster
  - Initially, choose $k$ of the points randomly to be centroids
  - Assign each point in the dataset to the closest cluster centroid
  - Recompute the cluster centroids to be the “means” of the points assigned to that cluster
  - Repeat till convergence

Guaranteed to converge, usually within a few iterations
Not optimal, finds a local minima (there are too many local minimas in the solution space)
Also somewhat sensitive to the initial choice of centroids

EM Algorithm: Expectation-Maximization Algorithm
Clustering

“k-means” algorithm

- At any point, have $k$ “centroids”, one for each cluster
- Initially, choose $k$ of the points randomly to be centroids
- Assign each point in the dataset to the closest cluster centroid
- Recompute the cluster centroids to be the “means” of the points assigned to that cluster
- Repeat till convergence

Guaranteed to converge, usually within a few iterations
Not optimal, finds a local minima (there are too many local minimas in the solution space)
Also somewhat sensitive to the initial choice of centroids
EM Algorithm: Expectation-Maximization Algorithm
Figure: K-Means Illustration (From Wikipedia Article)
 Algorithms like *k-means* not suitable for large datasets

- BIRCH provides a one-pass algorithm for clustering
- Supports a variety of distance metrics between clusters and across clusters

Key notions:

Given a set of points, $\vec{X}_i$, three terms capture all of their important properties:

$\text{CF} = (n, \sum \vec{X}_i, \sum \vec{X}_i^2)$

(Clustering feature)

Everything else (the distance metrics) can be computed using these, assuming you treat them as a unit

Easy to compute if two groups of points are merged

Also called "sufficient statistics"

There may exist distance metrics for which this won't hold (e.g. median-based metrics)
Algorithms like \textit{k-means} not suitable for large datasets

BIRCH provides a one-pass algorithm for clustering

Supports a variety of distance metrics between clusters and across clusters

Key notions:

Given a set of points, $\bar{X}_i$, three terms capture all of their important properties:

- $CF = (n, \Sigma \bar{X}_i, \Sigma \bar{X}_i^2)$ (Clustering feature)
- Everything else (the distance metrics) can be computed using these, assuming you treat them as a unit
- Easy to compute if two groups of points are merged
- Also called “sufficient statistics”

There may exist distance metrics for which this won’t hold (e.g. median-based metrics)
CF-Tree

Height-balanced Tree
A node contains at most $B$ entries of form $[CF_i, child_i]$
- $child_i$: pointer to the $i^{th}$ child
- $CF_i$: Clustering feature of all points grouped below $child_i$

Leaves: Same except that the diameter of the points must be below a threshold $T$
BIRCH: Scalable Clustering

- **CF-Tree**
  - Height-balanced Tree
  - A node contains at most $B$ entries of form $[CF_i, child_i]$
    - $child_i$: pointer to the $i^{th}$ child
    - $CF_i$: Clustering feature of all points grouped below $child_i$
  - Leaves: Same except that the diameter of the points must be below a threshold $T$

- **Basic Algorithm:**
  - Build the CF-Tree incrementally as the data is scanned
    - May have to adjust the threshold so that the tree fits in memory
  - Do a clustering using the final CF-Tree (using any clustering algorithm)
Points to note:
- Helps to imagine a “bottom-up” algorithm for clustering
  - Can think of each of the leaves as a “sub-cluster”
  - First we create fixed sized sub-clusters from the data (Phase 1)
  - Note: Leaf $\neq$ sub-cluster
  - Then do the clustering on the sub-clusters (Phase 3)
Points to note:

- Helps to imagine a “bottom-up” algorithm for clustering
  - Can think of each of the leaves as a “sub-cluster”
  - First we create fixed sized sub-clusters from the data (Phase 1)
  - Note: Leaf \( \neq \) sub-cluster
  - Then do the clustering on the sub-clusters (Phase 3)

- The CF-Tree has no “ordering” on the leaves
  - Same data point, if duplicated, may end up in different leaves

- The clustering at the sub-clusters is final
  - Once two points are put together in a single sub-cluster, no way to separate them
  - Since we have “lost” the original points
Points to note:

- Adjusting the threshold: requires very little extra memory
- Outliers:
  - Can cause problems
  - Each outlier must be in a separate sub-cluster
  - BIRCH handles this by spooling them to disk or throwing them out
Points to note:

- Adjusting the threshold: requires very little extra memory
- Outliers:
  - Can cause problems
  - Each outlier must be in a separate sub-cluster
  - BIRCH handles this by spooling them to disk or throwing them out
- How are the points assigned to the clusters?
  - At the end, we get a set of cluster centroids
  - Another pass is required to assign a "cluster-id" to the data points
  - Or can do it on demand (when the user asks for some point)
Classification

- Assign tuples to pre-defined labels (high/med/low “risk”)
- Supervised learning task: Usually given the labels for a set of tuples
- Usually based on some statistical model
- Commonly used techniques
  - Decision trees: A (usually) binary tree that you can traverse for each tuple
  - Bayesian classifiers (e.g. Naive Bayes)
  - Support vector machines etc.
- Much work on scaling these up in the Database literature
Learning decision trees:

- **Given**: A set of tuples and their classes
- **Goal**: Find the best decision tree of limited size
  - How to define “Best”? *Misclassifies fewest training tuples*
  - Why limited size?? Otherwise becomes too specific and not very useful (*overfitting*).

- NP-Hard problem.
Algorithm Overview:
- Find \((\text{attr}, \text{value})\) pair that differentiates between the labels in the training data the best
  - Different metrics: information gain, Gini index
- Set that as the root
- Split the dataset in two parts and recurse
- At the end, prune the decision tree found so that it is not very large
Algorithm Overview:
- Find \((\text{attr}, \text{value})\) pair that differentiates between the labels in the training data the best
  - Different metrics: information gain, Gini index
- Set that as the root
- Split the dataset in two parts and recurse
- At the end, prune the decision tree found so that it is not very large

Key algorithmic/scalability challenges:
1. Finding the best “split point” for very large training sets
   - Need to evaluate the information gain or Gini index many times
2. Splitting the dataset
3. How to parallelize
Aside: Evaluating Splits

- **Original tuples:** $S = (a, 1, \text{HIGH}), (a, 2, \text{HIGH}), (b, 1, \text{LOW}), (b, 2, \text{LOW})$
- **Schema:** (attr1, attr2, class label)

**Based on Gini Index Gain:**

- **Gini Index:** $G(p) = 1 - \sum p_i^2$, where $p$ is a distribution
  - Equals zero only if all tuples have the same class label
  - Here we use the distribution on the class labels
- $Gini(S) = 1 - 0.5^2 - 0.5^2$
- If $S$ split in $S_1$ and $S_2$, the gain =

$$Gini(S) - \left(\frac{|S_1|}{|S|} Gini(S_1) + \frac{|S_2|}{|S|} Gini(S_2)\right)$$

- Maximize this value over all possible splits (in this case, the split on attr1)
Aside: Evaluating Splits

- **Original tuples:** $S = (a, 1, HIGH), (a, 2, HIGH), (b, 1, LOW), (b, 2, LOW)$
- **Schema:** (attr1, attr2, class label)

**Based on Gini Index Gain:**
- **Gini Index:** $G(p) = 1 - \sum p_i^2$, where $p$ is a distribution
  - Equals zero only if all tuples have the same class label
  - Here we use the distribution on the class labels
- $Gini(S) = 1 - 0.5^2 - 0.5^2$
- If $S$ split in $S_1$ and $S_2$, the gain =
  \[
  Gini(S) - (\frac{|S_1|}{|S|} Gini(S_1) + \frac{|S_2|}{|S|} Gini(S_2))
  \]
  - Maximize this value over all possible splits (in this case, the split on attr1)

**Based on Information Gain:** Replace $G(p)$ with $H(p)$
SPRINT: A Scalable Parallel Algorithm

- Key ideas: Maintain several sorted lists
  - Can be easily split into two parts

Construct histograms to quickly compute the Gini index
  - Can be done with sorted attribute lists in one pass
  - Can be parallelized using “parallel sort” techniques
Outline

1. Overview
2. Association Rules
3. Clustering
4. Classification
   - Naive Bayes Classifier
5. Data Mining + Query Evaluation
Naive Bayes Classifiers

- Important class of classifiers to know about; simple; widely-used
- Given a data point, \( \vec{x} = \{x_1, \cdots, x_d\} \), the probability that it belongs to class \( c \) is given by:
  \[
p(class = c|\vec{x}) = \frac{p(c)p(\vec{x}|c)}{p(\vec{x})} \quad \text{(Bayes Rule)}
\]
Naive Bayes Classifiers

- Important class of classifiers to know about; simple; widely-used
- Given a data point, $\vec{x} = \{x_1, \cdots, x_d\}$, the probability that it belongs to class $c$ is given by:
  
  $$p(\text{class} = c | \vec{x}) = \frac{p(c)p(\vec{x} | c)}{p(\vec{x})} \text{ (Bayes Rule)}$$

  Where:
  
  - $p(c)$ = prob a random data point is in class $c$
  - $p(\vec{x} | c)$ = prob of $\vec{x}$ given the class label is $c$
  - $p(\vec{x})$ = probability of $\vec{x}$ (inconsequential)
Important class of classifiers to know about; simple; widely-used

Given a data point, $\vec{x} = \{x_1, \cdots, x_d\}$, the probability that it belongs to class $c$ is given by:

$$p(\text{class} = c | \vec{x}) = \frac{p(c)p(\vec{x} | c)}{p(\vec{x})} \quad \text{(Bayes Rule)}$$

Where:

- $p(c) = \text{prob a random data point is in class } c$
- $p(\vec{x} | c) = \text{prob of } \vec{x} \text{ given the class label is } c$
- $p(\vec{x}) = \text{probability of } \vec{x} \text{ (inconsequential)}$

To pick the class with the maximum $p(\text{class} = c | \vec{x})$, we instead pick the class with the maximum:

$$p(c)p(\vec{x} | c)$$
Important class of classifiers to know about; simple; widely-used

Given a data point, \( \vec{x} = \{x_1, \ldots, x_d\} \), the probability that it belongs to class \( c \) is given by:

\[
p(class = c | \vec{x}) = \frac{p(c)p(\vec{x} | c)}{p(\vec{x})} \quad \text{(Bayes Rule)}
\]

Where:

- \( p(c) = \text{prob a random data point is in class } c \)
- \( p(\vec{x} | c) = \text{prob of } \vec{x} \text{ given the class label is } c \)
- \( p(\vec{x}) = \text{probability of } \vec{x} \text{ (inconsequential)} \)

To pick the class with the maximum \( p(c | \vec{x}) \), we instead pick the class with the maximum:

\[
p(c)p(\vec{x} | c)
\]

\[
= p(c)p(x_1 | c)p(x_2 | c) \cdots p(x_d | c) \quad \text{(assume Independence)}
\]
Naive Bayes Classifiers

- We learn $p(x|c)$ for all $x$ and all $c$ from training data
- Also, we learn $p(c)$ for all $c$ from training data
- Note: Very few parameters $\implies$ not much training data required
We learn $p(x|c)$ for all $x$ and all $c$ from training data
Also, we learn $p(c)$ for all $c$ from training data
Note: Very few parameters $\implies$ not much training data required
Given a new data point $\vec{x}$:
- Compute $p(c)p(x_1|c)p(x_2|c) \cdots p(x_d|c)$
- Pick the class label with the largest value
1. Overview
2. Association Rules
3. Clustering
4. Classification
   - Naive Bayes Classifier
5. Data Mining + Query Evaluation
Data mining: Specialized algorithms
- Typically run outside the database
- The results might be added back in
- Not very efficient (recall MauveDB motivation)

Goal: Closer integration of mining models into a DBMS
Data mining: Specialized algorithms
   Typically run outside the database
   The results might be added back in
   Not very efficient (recall MauveDB motivation)

Goal: Closer integration of mining models into a DBMS
Specifically:
   Allow querying the “model output” as normal attributes
   “where risk = High” ("risk" is the predicted class label)
   Similarly in MauveDB, allow querying output of regression models
• Efficient Evaluation of Queries with Mining Predicates; Chaudhuri et al.
• Classification/Clustering Models
  • Allow querying the “class labels” directly
• Language Support?
  • MS Analysis Server allows defining classification models directly and allows querying
  • IBM Intelligent Miner (IM) also supports such functionality
  • Can think of these as “model-based views” as well
CREATE MINING MODEL Risk_Class
(
Customer_ID LONG KEY,
Gender TEXT DISCRETE,
Risk TEXT DISCRETE PREDICT,
Purchases DOUBLE DISCRETIZED(),
Age DOUBLE DISCRETIZED,
)
USING [Decision_Trees_101]

% Name of Model
% source column
% source column
% prediction column
% source column
% source column
% Mining Algorithm
CREATE MINING MODEL Risk_Class ( 
    Customer_ID LONG KEY, 
    Gender TEXT DISCRETE, 
    Risk TEXT DISCRETE PREDICT, 
    Purchases DOUBLE DISCRETIZED(), 
    Age DOUBLE DISCRETIZED, 
) 
USING [Decision_Trees_101] 

SELECT D.Customer_ID, M.Risk 
FROM [[Risk_Class] M 
PREDICTION JOIN 
(SELECT Customer_ID, Gender, Age, sum(Purchases) as SP 
    FROM Customers D Group BY Customer_ID, Gender, Age 
    ON M.Gender = D.Gender 
    and M.Age = D.Age 
    and M.Purchases = t.SP 
    Where M.Risk = ‘low’)
Query (abusing notation):
- select * from [Risk-Class] M where M.risk = "low"

Naive Approach
- Extract all rows that satisfy the rest of the predicates (none in this case)
- Apply the model to each row to predict the risk, and then check whether = "low"
Query (abusing notation):

- `select * from [Risk-Class] M where M.risk = "low"

Naive Approach

- Extract all rows that satisfy the rest of the predicates (none in this case)
- Apply the model to each row to predict the risk, and then check whether = "low"

Issues:

- Must do a seq scan and apply to all tuples (in this case)
- Can we do better?

Aside:

- Assumption that the model is to be applied for every tuple separately
Approximate Predicates

- Add predicates to the query that are guaranteed to not change the answer

Simple example:
  - `select * from R where R.a = 10`

Change to
  - `select * from R where R.a = 10 and R.a betn 9 and 11`
    - No use, but...
Approximate Predicates

- Add predicates to the query that are guaranteed to not change the answer
- Simple example:
  - select * from R where \( R.a = 10 \)
- Change to
  - select * from R where \( R.a = 10 \) and \( R.a \) betw 9 and 11
    - No use, but...
  - select * from R where \( R.a = 10 \) and \( R.b = 10 \)
    - IF it is known that \( R.a = R.b \) (maybe through a function dependency)
    - Faster to evaluate if there is an index on \( R.b \) (but not on \( R.a \))
    - Commonly done in most query optimizers
Similar ideas have been exploited before
- E.g. bounding rectangles in R+-Trees
Query optimizers infer new predicates to add to the given predicates
- Using transitivity, functional dependencies etc.
- Not approximate though
Approximate Predicates

- Similar ideas have been exploited before
  - E.g. bounding rectangles in R+-Trees
- Query optimizers infer new predicates to add to the given predicates
  - Using transitivity, functional dependencies etc.
    - Not approximate though
- In general, to add predicate $P_2$ to the query:
  - `select * from R where P1`
- We must have that:
  - $P_1 \implies P_2$
    - Not necessary the other way around
- Why?
  - $P_2$ might permit better access methods (indexes etc)
For a query: “label = 2”, find all leaves, find the predicate corresponding to it, and take a disjunction
For a query: “label = 2”, find all leaves, find the predicate corresponding to it, and take a disjunction

- (LowerBP > 91 and Age > 63 and Overweight = no) ∨
- (LowerBP > 91 and Age < 63) ∨
- (LowerBP <= 91 and UpperBP <= 130)
Consider a predicate: “label = 1”

Let $d$ be the number of attributes/dimensions

Imagine the $d$-dimensional hyperrectangle, where all points are marked as T or F

- T → “label = 1”
Consider a predicate: “label = 1”

Let \( d \) be the number of attributes/dimensions

Imagine the \( d \)-dimensional hyperrectangle, where all points are marked as T or F

\[ T \rightarrow \text{“label = 1”} \]

Cover all T’s using a set of hyperrectangles

Use them as the predicates

- Typically many ways of doing this
- Want to limit the number of rectangles
  - So could be approximate

If small No. of dimensions, could do this exhaustively

- Only needs to be done once for each label

But generally, must find more efficient ways
Figure: Mining Predicates: Creating a predicate for any classifier

Figure: Mining Predicates: Approximating the predicate constructed in previous figure
Mining Predicates: Any classifier...

**Figure:** Mining Predicates: Creating a predicate for any classifier

\[
R_a \in [1, 2] \text{ and } R_b \in [1, 2] \\
\text{or} \\
R_a \in [3, 4] \text{ and } R_b \in [1, 2, 3] \\
\text{or} \\
R_a = 3 \text{ and } R_b = 5
\]

**Figure:** Mining Predicates: Approximating the predicate constructed in previous figure

\[
R_a \in [1, 2] \text{ and } R_b \in [1, 2] \\
\text{or} \\
R_a \in [3, 4]
\]
Mining Predicates: Clustering

- Typically new points classified using their distance to the centroids
- The space gets split into convex polyhedrons
- Other types of clustering-based classifiers can be handled similarly, or using the algorithm for Naive Bayes
Mining Predicates: Clustering

- Typically new points classified using their distance to the centroids
- The space gets split into convex polyhedrons
- Other types of clustering-based classifiers can be handled similarly, or using the algorithm for Naive Bayes

**Figure:** Mining Predicates: Creating a predicate for a clustering-based classifier
Mining Predicates: Naive Bayes Classifiers

- Option 1: Enumerate the hyperspace and find the covering rectangles
- Option 2: Use the properties of the classifier to find the covering rectangles efficiently
- Can quickly check if a class $c$:
  - Holds for all the points in a hyperrectangle (MUST-WIN)
  - Holds for no point in a hyperrectangle (MUST-LOSE)
  - Holds for some points (AMBIGUOUS)
Mining Predicates: Naive Bayes Classifiers

- Option 1: Enumerate the hyperspace and find the covering rectangles
- Option 2: Use the properties of the classifier to find the covering rectangles efficiently

Can quickly check if a class \( c \):
- Holds for all the points in a hyperrectangle (MUST-WIN)
- Holds for no point in a hyperrectangle (MUST-LOSE)
- Holds for some points (AMBIGUOUS)

Start with the hyperrectangle covering the entire space
Subdivide until all hyperrectangles fall in one of the first two categories
A “threshold” parameter used to get an approximate but small set of rectangles
- In which case, some rectangles remain AMBIGUOUS
Can be extended to handle some other types of predicates as well.
Can be extended to handle some other types of predicates as well

Optimizers are not perfect
- May not like adding too many new predicates
- Too much complexity $\implies$ sequential scan
- Also selectivity estimates will get messed up
  - By definition, highly-correlated predicates

Very specific to the specific types of models being used
- How to generalize?