Computational Data Science
Fall 2018

Joel Huang

December 10, 2018

1 Entity Relationship Diagrams & SQL

Database Management Systems

Database Management Systems (DBMS) provide structured, efficient, and reliable storage and multi-user access to massive amounts of persistent data. To model database requirements, we need to first consider **conceptual design**: determine the entities and relationships, what data to store about them, and any other constraints that we want to impose. These will be used to construct the **entity relationship model**.

Entity-Relationship Models

- **Entities** are real-world objects, represented using *rectangles*.
- **Attributes** of entities are represented using *ovals*. They can have **domains**, which describe the data type of the attribute.
- **Relationships** are associations among multiple entities, represented using *lozenges/diamonds*. A one-to-two relationship may be called a ternary relationship set. Relationships can also contain attributes, and mapping constraints (e.g. many-to-many, etc.).
- The relationship **cardinality** is a measure of how much an entity participates in a relationship. A cardinality of $(1, x)$ implies mandatory participation of at least 1, up to x times, while $(0, y)$ implies optional participation of up to y times.

2 Data Handling in Unix

Previewing data

Raw data can be fairly large. We might want to examine the dataset's structure without opening it in a special program or text editor, which could take a long time and large amounts of memory.

Preview all - `cat`

Short for con**cat**enate. Sequentially reads file(s) and writes them to `stdout`. If redirection `>` is used, then output is written to the specified file. `>` is used to write to a file and `>>` is used to append to a file.

```
cat file1.txt
cat file1.txt file2.txt > newcombinedfile.txt
cat >newfile.txt
cat -n file1.txt file2.txt > newnumberedfile.txt
cat file1.txt >> file2.txt
cat file1.txt file2.txt file3.txt | sort > test4
```

Preview some - `head, tail`

Use `head` or `tail` to preview the head or tail of the data. Remember to supply the flags:

```
-n  Number of lines
-B  Display number of lines before
-A  Display number of lines after
```

Searching - `grep`

```
-E  Use extended regular expression syntax
-o  Output a matching segment of each line only
-n  Print the line number of each matched line
-C  Show a number of context lines too
```

```
.   Matches any character.
*   Matches zero or more instances of the
    preceding character.
+   Matches one or more instances of the
    preceding character.
[]  Matches any of the characters within the
    brackets.
```

```
()  Creates a sub-expression that can be combined
    to make more complicated expressions.
|   OR operator; (www|ftp) matches either 'www'
    or 'ftp'.
^   Matches the beginning of a line.
$   Matches the end of the line.
\   Escapes the following character. Since .
    matches any character, to match a literal
    period you would need to use \..
```

Replacing

3 Big Data, Hadoop, & MapReduce

Big Data

Big data involves challenges in raw **volume** of data, **variety** of sources, **velocity** of generation, **veracity** (trustworthiness) of data, and **value**.

Hadoop

Most real world data is semi-structured. Hadoop allows storing and manipulation of raw data to be reformatted later. This can be advantageous because pre-cleaned data can carry information that can be used in the future. Text, numerical data, and files can all be mixed.

CAP Theorem for Distributed Storage

It is impossible for a distributed data store to simultaneously provide more than two of the following:

1. **Consistency**, that every read operation will return the most recent write/an error,
2. **Availability**, that every request receives a non-error response (can be an outdated response),
3. **Partition tolerance**, that the system functions despite a communications break (drop or delay) between nodes.

Therefore, the possible compromises are:

- **C+A**, a single megacluster with nodes that always maintain contact, but will lose sync if there's a partition.
- **C+P**, where consistency is ensured throughout the network, but is not available when sync has not completed.
- **A+P**, where data is always returned despite a partition, but might not be accurate.

Big Idea: Mappers & Reducers

4 Counting Relative Frequencies

Big Idea: Approximate probabilities by counting occurrences in the data

k-Nearest Neighbours

A way to measure similarity between different data points, through determining similarity values or distances. Classification with k-NN is straightforward, but sensitive to the value of k, potentially computationally expensive (but can be sped up using kd-tree), and takes memory to store all data points.

Decision trees & Random Forests

Each feature represents an opportunity for branching. Decision trees are inexpensive and explainable, but prone overfitting withouot pruning To prevent overfitting, use Random forests-ensemble approach that aggregates decision trees's outputs via a majority voting system

Naive Bayes for classification

```
Assumption: Conditional independence
```

$$P(Y \mid x_1, x_2, \ldots, x_n) = P(Y \mid x_1) \cdot P(Y \mid x_2) \cdot \ldots \quad (1)$$

We are trying to find the most likely class given an observation Maximum Most likely class Y given feature set X

$$y_{MAP} = \arg\max P(Y \mid x_1, x_2, \ldots, x_m) \quad (2)$$

$$= \arg\max \frac{P(X \mid Y)\, P(Y)}{P(X)} \quad (3)$$

In equation (2), we're comparing the arg max of $P(Y = 0 \mid X)$ and $P(Y = 1 \mid X)$. Therefore we can cancel the constant denominator $P(X)$:

$$= \arg\max P(X \mid Y)\, P(Y) \quad (4)$$

Next, we need to find $P(X \mid Y)$ and $P(Y)$.

Count the prior probability $P(Y)$ Class labels are $Y \in \{0, 1\}$. Therefore, for a dataset with n labels,

$$P(Y = 0) = \frac{n_{Y=0}}{n} \quad (5)$$

$$P(Y = 1) = \frac{n_{Y=1}}{n} \quad (6)$$

In other words, there are $n_{Y=1}$ out of n occurrences of label Y having value 1.

Estimate $P(X \mid Y = 0)$ and $P(X \mid Y = 1)$ Since we have assumed conditional independence of X given classes Y (1),

$$P(x_i \mid Y = 0) = \frac{|x_{i,Y=0}|}{n_{Y=0}} \tag{7}$$

We are counting the probability (occurrences in the dataset) of feature x_i given $Y = 0$. Find all the rows where $Y = 0$ and count the occurrences of feature x_i. Repeat this for $Y = 1$.

Final comparison (arg max) Now we have pairs $P(Y = 0)$, $P(X \mid Y = 0)$, and $P(Y = 1)$, $P(X \mid Y = 1)$. Multiply the pairs as in equation (4) and take the *max* of the two.

Naive Bayes for word counting

```
Assumptions: Conditional independence,
    irrelevance of word order
```

We want to predict a class for a given sentence, for example, "good" for the sentence "love the burgers here, delicious and filling". We can estimate the class c using Naive Bayes,

$$c = \arg\max_{c_i \in C} P(c_i) \prod_{w_j \in W} P(w_j \mid c_i) \tag{8}$$

For a set of documents D, words W, and classes C, the probability of class c_i is simply the number of occurrences of c_i in document D.

$$P(c_i) = \frac{|c_i|}{|D|} \tag{9}$$

The probability of word w_1 given class c_i is also simply the number of occurrences of word w_1 in sentences with class c_i, divided by the probability of class c_i.

$$\begin{aligned} P(w_1 \mid c_i) &= \frac{P(w_1, c_i)}{P(c_i)} \\ &= \frac{count(w_1, c_i)}{\sum_{w_j \in W} count(w_j, c_i)} \end{aligned} \tag{10}$$

Example: Restaurant ratings

ID	Sentence	Class
1	The burger is **tasteless** and **slow** service	Bad
2	**slow** serving time and everything is **horrible**	Bad
3	Restaurant is near MRT, serves **delicious** burgers	Good
4	**love** the burger here, **delicious** and filling	Good
5	**love** this place, **delicious** burgers but **slow** service	?

Where classes $C = \{Bad, Good\}$, and words $W = \{tasteless, slow, horrible, delicious, love\}$.

$$P(c) = \begin{cases} \frac{2}{4}, & c = Bad \\ \frac{2}{4}, & c = Good \end{cases} \tag{11}$$

$$\begin{aligned} P(love \mid Good) &= \frac{count(love, Good)}{\sum_{w_j \in W} count(w_j, Good)} = \frac{1}{3} \\ P(delicious \mid Good) &= \frac{count(delicious, Good)}{\sum_{w_j \in W} count(w_j, Good)} = \frac{2}{3} \end{aligned} \tag{12}$$

$$\begin{aligned} P(Good \mid love, delicious) &= P(Good) \prod_{w_j \in W} P(w_j \mid Good) \\ &= P(Good) \cdot P(love \mid Good) \cdot P(delicious \mid Good) \\ &= \frac{2}{4} \times \frac{1}{3} \times \frac{2}{3} = \frac{1}{9} \end{aligned} \tag{13}$$

$$\begin{aligned} P(tasteless \mid Bad) &= \frac{count(tasteless, Bad)}{\sum_{w_j \in W} count(w_j, Bad)} = \frac{1}{4} \\ P(slow \mid Bad) &= \frac{count(slow, Bad)}{\sum_{w_j \in W} count(w_j, Bad)} = \frac{2}{4} \\ P(horrible \mid Bad) &= \frac{count(horrible, Bad)}{\sum_{w_j \in W} count(w_j, Bad)} = \frac{1}{4} \end{aligned} \tag{14}$$

Let's say we have the sentence $\{love, delicious\}$, and we want to find out the probability that its class label is *Bad*.

$$\begin{aligned} P(Bad \mid love, delicious) &= P(Bad) \prod_{w_j \in W} P(w_j \mid Bad) \\ &= P(Bad) \cdot P(love \mid Bad) \cdot P(delicious \mid Bad) \\ &= \frac{2}{4} \times \frac{0}{4} \times \frac{0}{4} = 0 \end{aligned} \tag{15}$$

Neither *love* nor *delicious* has appeared with the class *Bad* before. Therefore, we need smoothing to account for unseen words for specific classes.

LAPLACE SMOOTHING HERE. THEN FIX THE 0.

5 Feature Vectors

A feature is an individual, measurable property, or characteristic, of a phenomenon being observed.

Properties of features

We can compare features based on their properties:

Property	Operator	Example
1. Distinctness*	$=, \neq$	cat $\neq$ dog
2. Order of variables	$<, >, \leq, \geq$	A+ > B
3. Value comparison	$+, -$	+3°C difference
4. Ratio comparison	$\times, \div$	$A = 2 \times B$

*All features are distinct

Classes of features

Features may be binary, discrete, or continuous. In addition, they can be classified based on their properties. We can combine properties to give useful classes of features.

Class	Properties	Example
Nominal	1	Eye colors, postal codes
Ordinal	1, 2	Grades, race results
Interval	1, 2, 3	Dates, temperatures
Ratio	All	Distance, time

Feature Engineering

Feature engineering methods include aggregation, sampling, dimensionality reduction, feature subset selection, feature creation, discretization and binarization. In order to do this, we need to identify and solve the following problems when dealing with data.

Curse of dimensionality and Sparsity

When dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

Resolution and Quality

Patterns in data can depend on the scale of the features. Trends that might not be apparent in high-resolution data might appear in low-resolution data, and vice-versa. Quality of data can be affected by scale, inherent noise, and bad collection and sampling methods.

Removing noise and Detecting outliers

Noise is the random error (or variance) from inherently noisy observations or phenomena. **Outliers** are observations that are significantly atypical from most other observations in the dataset. We want to remove noise, so that our data becomes representative of reality (is this true?), and identify outliers (because the outlier events are *interesting*).

Missing values

We need to consider the reasons for missing values. For example, if the feature *annual income* is missing for a particular observation, it can be entirely random, or could be explained by another feature, say, *age*, where older adults tend to value privacy, intentionally not disclosing their income, or young children, who do not have annual incomes.

How do we deal with missing values? Based on the reasons we accept for the missing features, we can consider a few courses of action. If observations or features are missing completely at random (MCAR), due to data being randomly lost, we might choose to **eliminate** them. If the observations are missing in a time series, it might be possible to **estimate** the missing values. Alternatively, we might choose to **ignore** the missing value entirely, taking the rest of the observation without the value.

Duplicate data

Duplicate, or almost duplicate data, has to be taken care of. Heterogeneous sources, or independent collection sites, might produce them when merged into the final dataset. For example, one user can have multiple email addresses. Duplicate data has to be cleaned before the dataset can be used.

Wrong data

Features might be outright wrong. We might encounter negative values for weight, age, or entirely wrong addresses. We need to strategically reduce the probability of wrong data, by restricting user input values, input range, and automatic correction based on other features.

6 Clustering & Community Detection

Associations

Associations are useful to determine general trends in the database, with applications in **market basket analysis**. An itemset is a collection of one or more items. For a given transaction database T, we try to find interesting implications,

$$X \to Y, \tag{16}$$

given $X, Y \subset I$, where I is the itemset, and $X \cap Y = \varnothing$ (X and Y are mutually exclusive). For example, we might want to find and test the association rule $\{\, bread, butter \,\} \to \{\, milk \,\}$. This is the implication that a purchase of milk will occur together with the purchase of bread and butter. We can score these association rules based on the frequencies and combinations of the items.

Support count (σ), counts how many times the full itemset appears. The entire itemset has to be present in the transaction.

Support (s), is the fraction of transactions that contain an itemset. This is simply calculated by:

$$s(\{i_1, \ldots, i_n\}) = \frac{\sigma(\{i_1, \ldots, i_n\})}{|\, T \,|} \tag{17}$$

Frequent itemsets have a support value $\sigma \geq minsup$ ($minsup$ is a minimum support threshold we set to consider an itemset as frequent or not).

Scoring Association Rules

We need to find **association rules** of the form:

$$X = \{x_1, \ldots, x_n\} \to Y = \{y_1, \ldots, y_n\} \tag{18}$$

and we need to compare them to see if the relationships are strong. We introduce two metrics to score the rules. The **support** σ, of an association rule is the fraction of transactions that contain both itemsets X and Y, and **confidence** c, measures how often items in Y appear when transactions contain X.

$$\sigma = \frac{\sigma(X \cup Y)}{|\, T \,|} \tag{19}$$

$$c = \frac{\sigma(X \cup Y)}{\sigma(x)} \tag{20}$$

Mining Association Rules

The computational objective of mining for these rules is, to find all rules in T, with support $\sigma \geq minsup$, and $c \geq minconf$.

Brute force approach

The brute force approach will be computationally prohibitive. We need to list all possible rules, compute σ and c for each rule, and prune the rules that fail the $minsup$ and $minconf$ thresholds. In a database of d items, there are up to 2^d possible itemsets, and for each k-itemset of interest in T, the brute force approach costs $O(2^k)$, with $2^k - 2$ possible rules.

Apriori algorithm

```
Big Idea: Any subsets of a frequent itemset are
    also frequent itemsets.
```

Motivated by massive amounts of sales data with progress in bar-code technology circa 1994, Agrawal & Srikant from IBM Research presented the Apriori algorithm in *Fast Algorithms for Mining Association Rules*. The Apriori algorithm uses a bottom-up approach. Frequent subsets are extended one item at a time (candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori is historically significant but inefficient: the algorithm scans the database many times, while assuming its permanence in memory. Later algorithms try to identify maximal itemsets without enumerating their subsets.

The Apriori principle, stating that subsets of frequent itemsets are also frequent itemsets, holds because of the **anti-monotone** property,

$$X \subseteq Y \Rightarrow s(X) \geq s(Y). \tag{21}$$

This means that the support of an itemset never exceeds the support of its subsets. Therefore, if we

determine an itemset to be infrequent, we can automatically prune all parent sets (they are infrequent as well!).

Since we know the structure of the tree, we can first conduct a breadth-first search to find all 1-itemsets, then 2-itemsets, etc.