Tutorial #9. Relations

Definition

Let's once again consider Cartesian product. For 2 sets A and B – this is all possible ordered pairs, where first element is from A and second is from B. For 3 sets A, B and C Cartesian product is all possible triplets. And so on. We can extrapolate this to N sets: ordered combination is called **tuple**. In C#, for example, there is a class `Tuple` that implements this concept. Using concept of a tuple we can say, that Cartesian product is set of all possible tuples: **(a, b) ∈ AxB**, **(a, b, c) ∈ AxBxC, …**

Cartesian product is a *relation* "between everyone and everyone" *documented* as tuples in terms of sets. But if we want do describe more sophisticated relations, e.g. "x owns y", "x and y are parents of z", "salary of x is y"? These relations are true for only limited set of tuples, so to describe such relations we can remove all irrelevant tuples from Cartesian product!

ISPARENT(PARENT,KID)

Alice	Alice
Alice	Bob
Alice	Mom
Alice	Dad
Bob	Alice
Bob	Bob
Bob	Mom
Bob	Dad
Mom	**Alice**
Mom	**Bob**
Mom	Mom
Mom	Dad
Dad	**Alice**
Dad	**Bob**
Dad	Mom
Dad	Dad

BRO_OR_SISTER(A,B)

Alice	Alice
Alice	**Bob**
Alice	Mom
Alice	Dad
Bob	**Alice**
Bob	Bob
Bob	Mom
Bob	Dad
Mom	Alice
Mom	Bob
Mom	Mom
Mom	Dad
Dad	Alice
Dad	Bob
Dad	Mom
Dad	Dad

Any **subset** on Cartesian product is a **relation**! **R⊆ AxB**. To check whether relation between ***a*** *and* ***b*** is valid or not we have to find whether **(a,b) ∈ ℝ.** Or, if we want shorter form: **a R b**. Sounds familiar? Yes! Because **function is also relation**. But relation with constraints:

- $\forall x \in Xdomain\ \exists y \in Ycodomain\ (x, y) \in F$
- $\forall x \in Xdomain\ \exists! y \in Ycodomain\ (x, y) \in F$

Properties

We remember the following properties of functions (and relations in common):

- Injective
- Subjective
- Bijective

Same as functions all relations have **inverse of relation**: just switching elements in a pair. Unlikely the functions, relations have no constraints on inverse.

$$\boldsymbol{R^{-1} = \{(y, x) \in (BxA) | (x, y) \in R\}}$$

Relations also have other restriction properties:

- The simplest relations – **binary** relations. They consider relations between just 2 sets. There can also be *n*-ary relations, they can easily be imagined as a table in RDBMS.
- Binary relations can be OVER some set. Then they can be specified using Cartesian power: R⊆AxA. For such kinds of relations we can consider following properties:
    - **Reflexive**. Means **x ∈ A, we have x R x (or (x, x) ∈ R)**
        - There is a difference between non-reflexive and irreflexive (anti-reflexive relation)
            - **Non-reflexive**: $\exists x, (\boldsymbol{x}, \boldsymbol{x}) \notin \boldsymbol{R}$
            - **Irreflexive**: $\forall x, (\boldsymbol{x}, \boldsymbol{x}) \notin \boldsymbol{R}$
    - **Transitive**. Means **(x R y) & (y R z) → (x R z)**
    - **Symmetric.** Means **(x R y) → (y R x)**
        - **Ask students to provide examples of such relations**
        - There is a difference between non-symmetric, asymmetric and antisymmetic relations
            - **Non-symmetric**: $\exists x, y$: $(\boldsymbol{x}\ \boldsymbol{R}\ \boldsymbol{y})\ \wedge \neg(\boldsymbol{y}\ \boldsymbol{R}\ \boldsymbol{x})$
            - **antisymmetric**: $\forall x, y$: $(\boldsymbol{x}\ \boldsymbol{R}\ \boldsymbol{y})\ \wedge (\boldsymbol{y}\ \boldsymbol{R}\ \boldsymbol{x}) \rightarrow (\boldsymbol{x} = \boldsymbol{y})$
            - **asymmetric** = anti+irreflexive: $\forall x, y$: $(\boldsymbol{x}\ \boldsymbol{R}\ \boldsymbol{y})\ \rightarrow \neg(\boldsymbol{y}\ \boldsymbol{R}\ \boldsymbol{x})$
- If we combine **strictly [ir]reflexive, transitive and antisymmetric** properties: this relation will be **partial order relation**. It can be either:
    - Weak: strictly **irreflexive (<=)**
    - Strict: strictly **reflexive (<)**
- If we combine **reflexive, transitive and symmetric** together, than we get **EQUIVALENCE relation**. Or, simpler, all elements in some group are somehow similar to each other. (Ask students to provide examples of such relations). Very important point about equivalence is that this relations split sets into **disjoint subsets**. E.g. “same color” of “same whatever” are equivalence relations – they cannot intersect.

Clustering or partitioning

Clustering is a very important practical task. If there’s too much data to process – how can we make our life easier considering large groups of such objects as something similar? Or in other words how to find equivalence classes? $[\boldsymbol{x}]_{\boldsymbol{R}} = \{\ \boldsymbol{y} \in \boldsymbol{A}\ |\ \boldsymbol{xRy}\ \}$. Or even simpler – how to solve clustering task?

E.g. Parallels has 4M users, some of them face problems. These problems convert to 30GB of report data every day. Problems are usually caused by bugs and come together with stack traces. Stack traces for different versions of OS, versions of product or even different software installed can be different for the same problem. How can we find equivalence classes $[\boldsymbol{bug\_report}]_{\boldsymbol{same\_bug}}$ *for the problem? Here comes ML approaches! We need to find a* ***feature*** *that will be the same for different reports related to the same bug. This feature is called exception signature and is calculated like “module_classname_method_name_offset”. This feature in practice allows to aggregate hundreds of different exception stack traces to a single JIRA bug.*

Consider with students other examples of useful clustering.

Let’s consider 2 ways of finding features (or in other words, find something similar):

1) intuition about the data + heuristics
2) feature selection with ML.

In first case we understand the relation that lies under equivalence class, in second – we cannot imagine them.

Let's try to restore your knowledge from summer school and use K-means algorithm for grades classification to A-D given raw grading data.

- What kind of feature selection will be used? (1st, because we have intuition on number of equivalence classes and distance metrics).
- What will be the relation for class equivalence? What is the feature? (elements are closer to the same centroid $[x]_R$ according to the distance metrics provided by user).

Write or adopt k-means algorithm for grade distribution. Suggested metrics: final sum, or weighted sum according to question average value.

Reference materials: *page 13. 1.3. Language of relations and functions*