Using graphs for feature engineering pipelines.

Wes Madrigal

ODSC East 2024

ML/AI on tabular data

P(purchase | transactions, products, customer)

(a) Relational Tables **(b)** Define Tasks

Models need flat tables not graphs

ML needs this: [1, 6, 33.3, ‘product 5’, ‘opened notification’]

Not this:

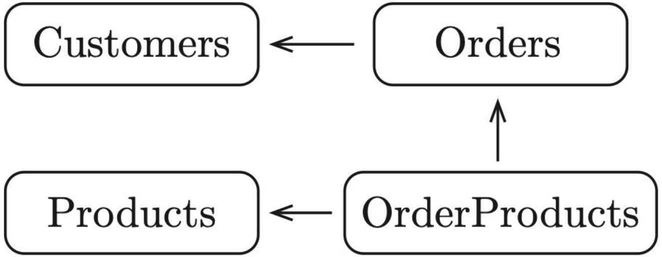
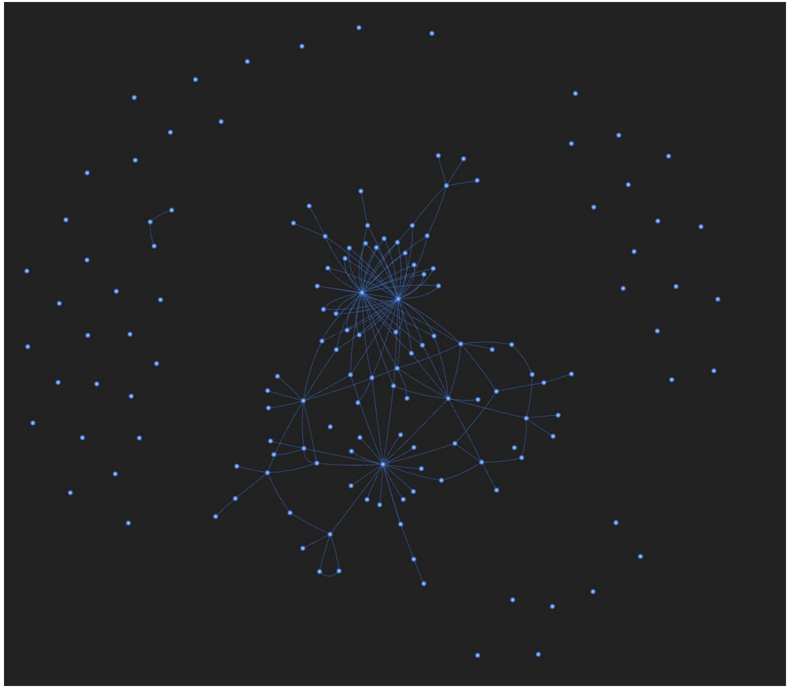


Fig. 2. A simplified schema for an e-commerce website. There are 4 entities. An arrow from one entity to another signifies that the first entity references the second in the database.

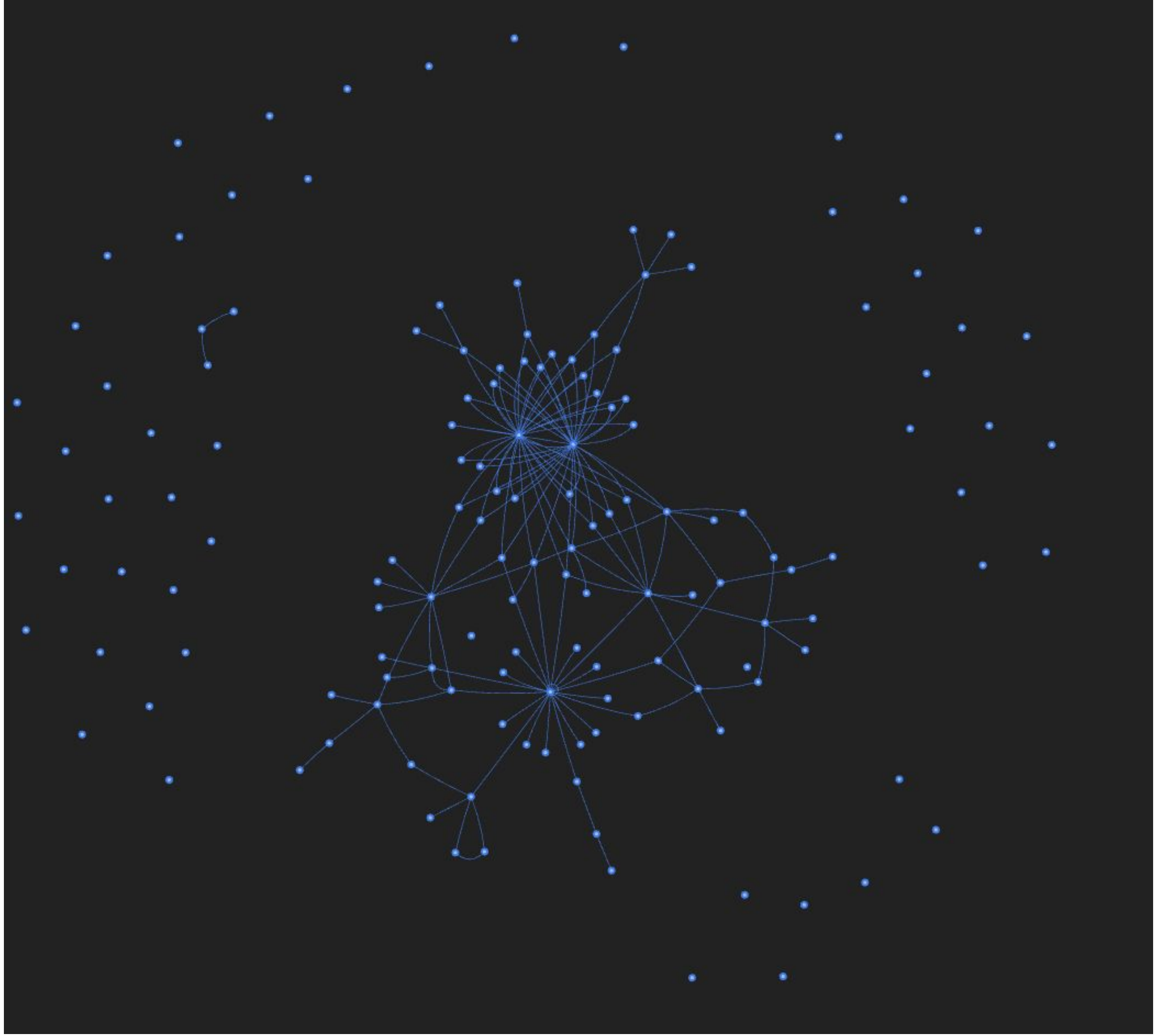


Actual customer Entity graph

Ideally we have:

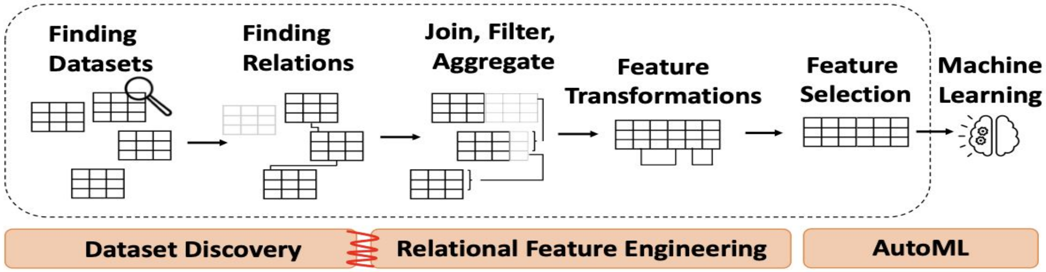
P(target | entire database)

Conditioning a target on an entire database is, however, a challenge.



Problem

- Much of world's most valuable data is stored in tabular warehouses/lakes where data is spread across many tables/files.
- To date no ML/AI method can learn directly on disparate tabular data
- ML/AI methods require a single "flat" table, which is achieved through the process of feature engineering
- Feature engineering is where data scientists spend a lot of time
- *Without a reusable, composable interface for building and automating features technical debt and system complexity increases.*



Why does feature engineering complexity matter?

- The failure rate of AI projects is high (85%), therefore experiment speed matters.
  - https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence
- The cost of AI projects is high, therefore the reusability, extensibility, and production readiness of high importance.
  - https://www.phdata.io/blog/what-is-the-cost-to-deploy-and-maintain-a-machine-learning-model/
  - Bare bones without MLOps: $60K
  - With MLOps for 1 model: $95K
- The talent shortage exacerbates the aforementioned
  - https://www.forbes.com/sites/forbestechcouncil/2022/10/11/the-data-science-talent-gap-why-it-exists-and-what-businesses-can-do-about-it/?sh=3c63f6f23982

- Summary: If you don’t care, your boss does. If they don’t care, their boss does

Prior work

Papers:

- Deep Feature Synthesis: https://www.maxkanter.com/papers/DSAA_DSM_2015.pdf
- One Button Machine (IBM): https://arxiv.org/pdf/1706.00327.pdf
- Autofeat (BASF): https://arxiv.org/pdf/1901.07329.pdf
- TSFresh: https://arxiv.org/pdf/1610.07717.pdf

Code/projects:

- Featuretools: https://github.com/alteryx/featuretools
- AutoFeat: https://github.com/cod3licious/autofeat
- TSFresh: https://github.com/blue-yonder/tsfresh
- FeatureSelector: https://github.com/WillKoehrsen/feature-selector

Companies:

- dotData
- Alteryx
- Trifacta
- Kumo.ai (using GNNs)

Shortcomings of prior research

- Extensibility is lacking
  - Point in time correctness is only handled in 1-2 of the implementations, but is crucial.
  - Deep Feature Synthesis requires pandas dataframes
  - Alteryx featuretools requires pandas dataframes (Spark is in Beta)
  - One button machine (IBM) uses Spark but their implementation could not be found
  - TSFresh operates on single files only
- Customizability is non-existent
  - *Most* of the papers depend on local compute engine, such as pandas
  - This makes bringing in third-party libraries, such as cleanlab, a challenge if not impossible

How do I update, modify, and maintain this?

```
select c.id as customer_id, nots.num_notifications,
nots.total_interactions,
nots.avg_interactions,
nots.max_interactions,
nots.min_interactions
from customers c
left join
(
select n.customer_id, count(n.id) as num_notifications, sum(ni.num_interactions) as
total_interactions,
avg(ni.num_interactions) as avg_interactions,
max(ni.num_interactions) as max_interactions,
min(ni.num_interactions) as min_interactions
from notifications n
left join
(
select notification_id,
count(id) as num_interactions
from notification_interactions
group by notification_id
) ni
on n.id = ni.notification_id
group by n.customer_id
) nots
on c.id = nots.customer_id
left join
(
select o.id as order_id, o.customer_id,
oe.num_order_events,
oe.num_type_events
from orders o
left join
(
select order_id,
count(id) as num_order_events,
sum(case when event_type_id = 1 then 1 else 0 end) as
num_type_events
from order_events
group by order_id
) oe
on o.id = oe.order_id
left join (
select order_id,
count(id) as num_order_products,
sum(case when product_type_id = 5 then 1 else 0 end) as
num_expensive_products,
sum(product_price) as product_price_sum,
max(product_price)-min(product_price) as product_price_range
from order_products
group by order_id
) op
on o.id = op.order_id
) ods
on c.id = ods.customer_id
where c.is_high_value = 1
and c.is_test = 0
and c.some_other_filter = 'yes';
```

What about orientation in time?

WHERE some_col >= ‘YYYY-MM-DD’ AND some_col < ‘YYYY-MM-DD’

…

…

```sql
select order_id,
count(id) as num_order_products,
sum(case when product_type_id = 5 then 1 else 0 end) as num_expensive_products,
sum(product_price) as product_price_sum,
max(product_price)-min(product_price) as product_price_range
from order_products
where ts > '2023-01-01' and ts < '2023-05-01'
```

Python: df.filter[df[col] >= ‘YYYY-MM-DD’ AND some_col < ‘YYYY-MM-DD’]

How about cardinality?

- Customer
  - N = 1,000
- Orders
  - 20N = 20,000
- Order Events
  - 200N = 200,000
- Order Products
  - 50N = 50,0000
- Notifications
  - 200N = 200,000
- Notification Interactions
  - 600N = 600,000

```sql
select c.id as customer_id,
o.id as order_id,
o.amount as order_amount,
oe.event_type as order_event_type,
n.id as notification_id
from customers c
left join orders o
on c.id = o.customer_id
left join order_event oe
on o.id = oe.order_id
left join notifications n
on c.id = n.customer_id;
```

Returned rows = ~40,000,000

Solution

- GraphReduce is a programming model and associated software abstractions.
- The associated abstractions handle redundant logic such as joins, date filtering, certain aggregations, etc.
- Top-level parameters such as dates, data consideration windows, target windows, etc. are all front loaded.
- Need the following:
    - Ability to switch parent / root table
    - Orientation in time across entire graph
    - Abstractions for repetitively implemented logic, such as joins, group bys, filters, etc.
    - Support multiple feature definitions for same table
    - Production-ready (scales to large data)
    - Interoperable across compute layers (pandas, dask, spark, ray, snowflake, databricks)

```
gr = GraphReduce(
    name='notifications',
    parent_node=gr_nodes['cust.csv'],
    fmt='csv',
    cut_date=datetime.datetime(2023,9,1),
    compute_layer=GraphReduceComputeLayerEnum.pandas,
    auto_features=True,
    auto_feature_hops_front=1,
    auto_feature_hops_back=2,
    label_node=gr_nodes['orders.csv'],
    label_operation='count',
    label_field='id',
    label_period_val=60,
    label_period_unit=PeriodUnit.day
)
```

Assumptions:

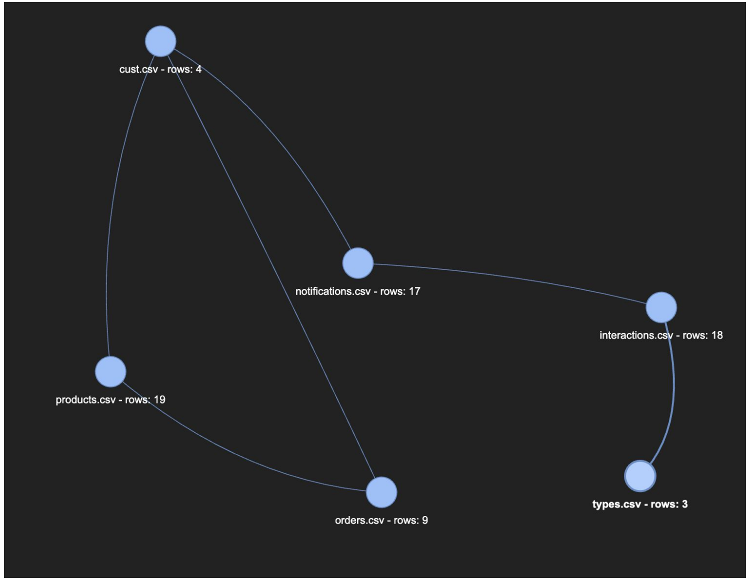
- Does not need feature searchability
- Does not need a managed storage layer, you can probably manage S3/Blob/GCS yourself?
- Does not roll an orchestrator
- Does not handle interop and mapping between online feature definitions and batch ones

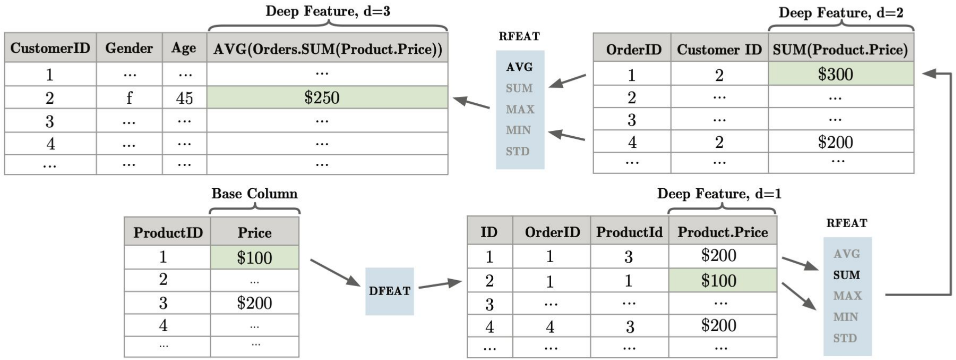
Solution continued…

- Graphs can serve as the data structure for this problem by representing tables as nodes and foreign keys as edges.

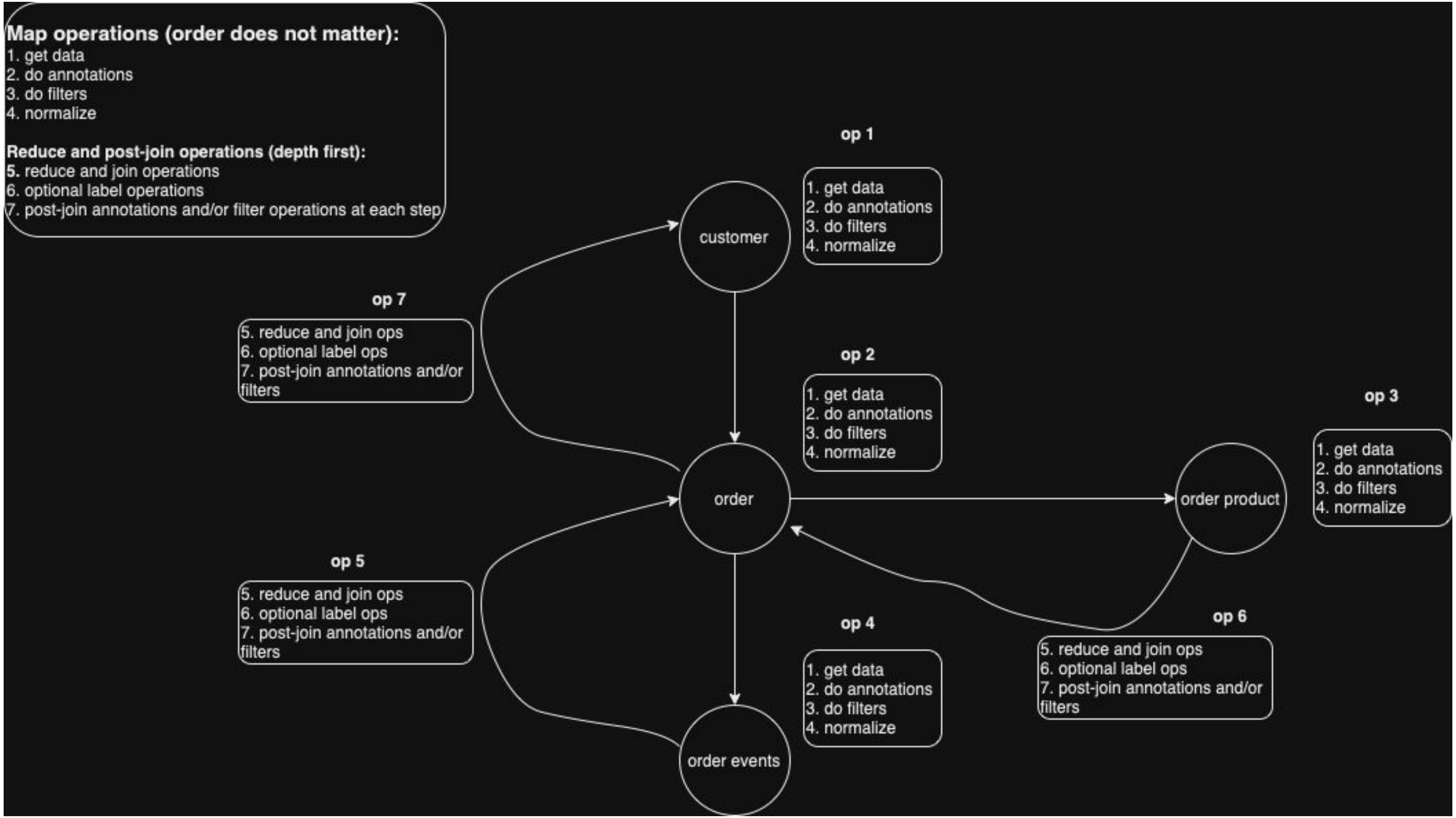
- By leveraging graph data structures we can plug into existing open source:
  - https://github.com/networkx
  - https://github.com/WestHealth/pyvis

- Some other companies have taken this approach with GNNs
  - https://kumo.ai





Operations diagram



GraphReduce

- **GraphReduce**
  - Top-level class that subclasses **nx.DiGraph** and defines abstractions for
  - Cut dates: the data around which to orient the data
  - Consideration period: the amount of time to consider
  - Compute layer: the compute layer to use
  - Abstractions for enforcing naming conventions and sequence
  - Edges between nodes and edge metadata (e.g., cardinality between nodes)
  - Compute graph specifications, such as whether to reduce a node or not

- **GraphReduceNode**
  - Custom class for each node, which allows parameterization of the following:
    - Primary key
    - Date key
    - File path
    - File format
    - Compute layer
    - prefix

DEMO

Visit: https://bit.ly/odsc

Next steps

- **P(target | entire database)**

- Reducing boilerplate code required
- Better metadata management (around features)
- More feature primitives based on semantic types
- Automatic inference of time units
- Enhancements to visualization, graph serialization, and tracking
- Integration with other projects
  - Cleanlab
  - Others