Discovering Customer Signals in Online Retail

JON VINES
Customer Signals / Interaction Events
Setting the scene
A little about me

Jon Vines

Engineering Lead, AO.com

Co-organizer, DevOps Manchester
Introducing AO

Online electrical retailer in UK and Germany

Operate across MDA, SDA, AV and other categories

Recycle over a quarter of the UK's total fridges per year
Consumer Expectations

1. Companies understand my wants and needs
2. Adapt based on their actions and behaviour
3. Happy with current level of personalisation
4. Willing to share data if it means a cheaper and easier experience
Consumer Expectations

---

Discovering Customer Signals
Enabling personalisation

Understand where our data is coming from

Give our data context, what else is happening?

Make data available across the organisation

Use data in real-time
Gathering our thoughts
Data is central to the single customer view that organisations require to better understand their customers and tailor hyper-personalised engagement.
Events over state

Databases represent state at that point in time

Events represent an action at a point in time
What is an Event

An event is something that happens, especially when it is unusual or important.
Sources of Events

- Order confirmation
- Delivery Van Location
- System Event
Types of Events

- Event Notification
- Event Carried State Transfer
- Event Sourcing
Customer Signals

Discovering Customer Signals
Looking at the data
Interaction Events

- Capture every interaction our visitors have on site
- Capture enough context for us to understand what they’ve done
- Analyse for trends at a macro level
- Analyse for behaviours at a micro level
- Respond to emerging behaviours for better customer experience
Storing Interactions Events

Document per interaction

Document for all interactions

Buckets with capped documents
Capped Documents

- Size limit to document model
- Exceed size limit, create new document
- Resource efficient
- Suited to irregular updates
- Prevents out of bound growth
- Large number of documents
Buckets

- Identify the document we wish to update

- Determine a cap for the number of entries into the array we want to cap by

```csharp
var sessionIdFilter = Builders<SessionInteractions>.Filter.Eq(x => x.SessionId, sessionId);
var interactionIdFilter =
    Builders<SessionInteractions>.Filter.Lt(x => x.InteractionCount, BucketSize);

var filter = Builders<SessionInteractions>.Filter.And(sessionIdFilter, interactionIdFilter);

var update = Builders<SessionInteractions>
    .Update
    .SetOnInsert(x => x.SessionId, sessionId)
    .SetOnInsert(x => x.VisitorId, visitorId)
    .SetOnInsert(x => x.DateInserted, DateTime.UtcNow)
    .Set(x => x.LastUpdated, DateTime.UtcNow)
    .AddToSet(x => x.Interactions, new Interaction(DateTime.UtcNow, eventType, payload, tags))
    .Inc(x => x.InteractionCount, 1);

var result = _database.GetCollection<SessionInteractions>(_collectionName)
    .UpdateOne(filter, update, new UpdateOptions {IsUpsert = true});
```
Buckets

- Update values on creation only
- Update on every interaction
- Increment the count

```csharp
var sessionIdFilter = Builders<SessionInteractions>.Filter.Eq(x => x.SessionId, sessionId);
var interactionIdFilter = Builders<SessionInteractions>.Filter.Lt(x => x.InteractionCount, BucketSize);

var filter = Builders<SessionInteractions>.Filter.And(sessionIdFilter, interactionIdFilter);

var update = Builders<SessionInteractions>
    .Update
    .SetOnInsert(x => x.SessionId, sessionId)
    .SetOnInsert(x => x.VisitorId, visitorId)
    .SetOnInsert(x => x.DateInserted, DateTime.UtcNow)
    .Set(x => x.LastUpdated, DateTime.UtcNow)
    .AddToSet(x => x.Interactions, new Interaction(DateTime.UtcNow, eventType, payload, tags))
    .Inc(x => x.InteractionCount, 1);

var result = _database.GetCollection<SessionInteractions>(_collectionName)
    .UpdateOne(filter, update, new UpdateOptions { IsUpsert = true });
```
Buckets

- Combine filter and update
- Set IsUpsert to true

```csharp
var sessionIdFilter = Builders<SessionInteractions>.Filter.Eq(x => x.SessionId, sessionId);
var interactionIdFilter =
    Builders<SessionInteractions>.Filter.Lt(x => x.InteractionCount, BucketSize);

var filter = Builders<SessionInteractions>.Filter.And(sessionIdFilter, interactionIdFilter);

var update = Builders<SessionInteractions>
    .Update
    .SetOnInsert(x => x.SessionId, sessionId)
    .SetOnInsert(x => x.VisitorId, visitorId)
    .SetOnInsert(x => x.DateInserted, DateTime.UtcNow)
    .Set(x => x.LastUpdated, DateTime.UtcNow)
    .AddToSet(x => x.Interactions, new Interaction(DateTime.UtcNow, eventType, payload, tags))
    .Inc(x => x.InteractionCount, 1);

var result = _database.GetCollection<SessionInteractions>(_collectionName)
    .UpdateOne(filter, update, new UpdateOptions {IsUpsert = true});
```
Aggregation Framework

- We want to see an aggregated view of the session
- Use aggregation framework to create this view
- Publish an aggregated session view for downstream use
Machine Learning Models

- Session stats stored in S3 for training
- Define and extract features
- Extend session stats with features
- Use same model for online execution
Atlas Data Lake

- MongoDB query language over S3
- Data exploration
- Same query for offline exploration and online execution
- Close eye – still in beta
Monitoring Effectiveness

- Confusion matrix
- Aggregates model accuracy
- Time-boxed aggregations of effectiveness
- See results in real-time
Plotting our next steps
MongoDB and Events

- Structure of data in MongoDB closely matches structure of events
- Monitor the inserts, updates and deletes to generate events
- Keep this in mind when designing event-driven service
Change Streams

- Access real-time data changes
- Subscribe to all data changes on a single collection, a database or deployment
- Filter for specific changes or transform the notifications

```javascript
var cursor = inventory.Watch();
while (cursor.MoveNext() && cursor.Current.Count() == 0) { }
var next = cursor.Current.First();
cursor.Dispose();
```
Kafka Connect

- We already use Kafka
- Kafka Connect is a framework for connecting Kafka with external systems
- Fully supported Kafka Connect for MongoDB with sink and source connectivity

```bash
curl -X PUT http://localhost:8083/connectors/source-mongodb-inventory/config \
   -H "Content-Type: application/json" -d '{
   "tasks.max":1,
   "connector.class":"com.mongodb.kafka.connect.MongoSourceConnector",
   "key.converter":"org.apache.kafka.connect.storage.StringConverter",
   "value.converter":"org.apache.kafka.connect.storage.StringConverter",
   "connection.uri": "<>",
   "database": "Clickstream",
   "collection": "interactions",
   "pipeline": "[{}]",
   "topic.prefix": ""
}
'
Use Cases

Discovering Customer Signals
Results

Indicative increase in sales growth

Reduced model delivery time

Monitor real-time results
In Summary

- Consumers expect a personalised experience
- Consumers are consistently sending signals for us to adapt to and understand
- Embrace events to decouple systems and create opportunities
- Reduce the disconnect between offline training and online execution
Customer Signals

Discovering Customer Signals
Customer Signals
Thank You

Twitter @jonvines    LinkedIn jonvines