Everything is Search

Maxim Noah Khailo

January 23, 2016

1 What is Dynamic Website Display Really?

Whenever you display something on a web page with dynamic content, it requires search. The typical case is searching by id, or within some time range. These are what are called precise queries and the results are typically exactly accurate. Or perfect matches.

- But why should the queries be precise?
- Why can't the system figure out which appropriate image or which description to show at any moment?
- And why does the result have to be one item and not a set?

This article attempts to answer these questions and create a novel model for storing and retrieving content such as product descriptions, image references, recommendations, etc. Automatically choosing the most appropriate item to display based on user feedback.

2 Search and Selection

There are two fundamental ideas that I will attempt to generalize and combine in an interesting way.

2.1 Search

Any dynamic website display requires search. Generally this is achieved by a precise search represented as a predicate. A predicate on a set of objects creates a subset.

Example predicates might be...

- Object with id X.
- Objects between time T_1 and T_2.

The first insight is that search does not have to be precise and can take a different form. One such form is is a spatial N-Nearest Neighbors search. This type of search answers a different sort of query.

- N Items that are similar to X.

And instead of using a predicate, we use the concept of 'close' which is defined by a distance function.

2.2 Selection

Generally a search is done on a set of objects such as products, customers, images, etc. The second generalization is that, instead of searching on a set of objects, we can search on a set of sets instead.

- Customer Groups instead of Customers.
- Product Representations instead of Products.

This derivative approach requires a second algorithm called "Selection". Selection is a function that takes a set and chooses and object from that set based on some criteria.

1. Given subset G of sets after a search.
2. Select object O from each set in G for display.

This is powerful because it means we can provide multiple representations of a product (multiple images, descriptions, and layouts) and can have some function to choose which one is most appropriate to display. I will provide one such selection algorithm later here.

3 Spatial Search

3.1 Representing Information as Multi-Dimensional Vectors

Most information can be summarized using a multi-dimensional vector called a feature vector. Feature vectors are multi-dimensional numerical representations of some kind interesting information.

$$< 1, 5, 23, ..., n > \tag{1}$$

The vector above stores each component as a number and each component belongs to a dimension. For example you can store a 2D point as a vector

$$< x, y > \tag{2}$$

Where the first component is the x dimension, and the second is the y dimension.

Any piece of information can be mapped to a vector with N dimensions. Mappings can be exact or lossy. We can attempt to map a product to a vector like so

$$< id > \tag{3}$$

Above is a vector of one dimension where the product id is used as the value. This is what most databases do when you search for ID. They store the ID in some spatial data structure and retrieve a pointer to the actual data.

We can also store another mapping of a product like this.

$$< language, region, modality, category, id > \tag{4}$$

Where we assign a numerical value to each item. This can also be stored in a spatial data structure like a KD-Tree.

3.2 Defining Distance

Lets go back to our previous example of the 2D point.

$$< x, y > \tag{5}$$

We can compute how far apart two 2D points are by subtracting the vectors and and computing the distance via Pythagorean Theorem.

$$distance(< x_1, y_1 >, < x_2, y_2 >) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{6}$$

The beauty of the Pythagorean Theorem is that it works with vectors of arbitrary dimensions.

This means we can take any feature vector and compute how close two of them are. So if we take our previous product representation minus the id...

$$< language, region, modality, category > \tag{7}$$

We can compute the distance between two products

$$distance = \sqrt{(language_1 - language_2)^2 + ... + (category_1 - category_2)^2} \tag{8}$$

There are many data structures which can compute N-Nearest Neighbors like KD-Tree or cover tree.

3.3 Other Distance Functions

Pythagorean theorem is one approach to computing distance. Distance can also be computed using other functions...

- Projected vector to the surface of a Hyper-Sphere and use Cosine Distance.
- Compute an approximation of distance using Hamming Distance.
- If the vector represents text, you can use an edit distance algorithm like Wagner-Fischer.

3.4 Clustering

The advantage of creating feature vectors is that they can be stored in a spatial data structure and queried. Since the feature vectors can store a lossy representation of the information, you can perform fuzzy searches and cluster similar objects together.

For example, you can cluster information via K-Means. We can store the vectors in something like a KD-Tree or Cover Tree to speedup N-Nearest Neighbor search.

4 Dynamic Selection

Instead of an object, we can attach a set of objects instead to our feature vectors. This means we need a selection algorithm after our search to choose what to display. Why not just point to the object we want to display directly? Because we don't actually know what we want to display!

For example, what if you have multiple images or descriptions for a product. Which one is best to display? Even harder, which one is better to display on the storefront vs mobile vs a 3rd party like Amazon or Ebay?

We need to answer this kind of question with data and a selection algorithm.

4.1 The Multi-Armed Bandit Problem

We can formalize our selection problem as the multi-armed bandit problem. That is the expected "payout" from each object in the set has yet to be determined and we want a strategy for choosing one with the highest payout as we accumulate more data.

For example, we don't know which image, or product description is most appropriate to display. The multi-armed bandit model is an attempt to address this by having an adaptive algorithm that accumulates data and chooses the most appropriate object with the most payout. Where payout here can be clicks over impressions.

There are usually two phases in the approach called the exploratory phase and the exploitation phase. One of the most simplest strategies to tackling these phases is called the Epsilon-Greedy strategy.

This is where for some percentage, you choose the object with the highest payout, and for the rest, you pick at random.

4.2 Computing Payout

Determining payout requires counting. For a website that normally means counting impressions and clicks, where clicks are considered a payout. The object with the highest payout is computed by taking clicks over impressions.

$$payout = clicks/impressions \tag{9}$$

Given our feature vector pointing to a set of objects. Each object needs to have statistics attached to it. We can store simple counts but that doesn't provide valuable information for insights. What we need to do is store these values in a time series database. It is also preferable that we store the integrated value, "the total", in this time series database because we can compute changes over time using simple subtraction between times.

$$\begin{aligned} clicks &= clicks[t_1] - clicks[t_2] \\ impresions &= impressions[t_1] - impressions[t_2] \\ payout &= clicks/impressions \end{aligned}$$

Having a time series database will also allow us to provide graphs for insights and demonstrate the system's effectiveness at improving payout.

For example, if we store the clicks and impressions on the set that the feature vector points to, and not just the objects of the set, then we can graph the improvements in payout when our system makes decisions on what to display. We can graph the points when our algorithm chooses to switch display and then we should see an improvement in payout over time.

4.3 Contextualized Bandit

There is a version of Multi-Armed Bandit that uses a context feature vector to help drive deciding what to choose. Given my idea of search and selection, using feature vectors pointing to sets of objects, we can easily implement fancier algorithms in the future since we will essentially have a context vector.

This includes training linear classifiers, neural nets, etc.

5 What does MVP look like?

This system sounds like a lot of work to build, and it is. However we can build it in a lean way, just-in-time, when customers want it by having the right architecture.

MVP to me looks like a normal content system with the following generalizations

5.1 Have a place to Switch Index Based on Feature Vector

In an MVP, each object has a feature vector with one value, which has the id of the object. We then have a place in our system to change which index we search with the feature vector. In the beginning, this function would just point to ES or Postgresql and search for id as a normal content system would.

5.2 Have a Set of One

Each index has a node object that looks like this.

```
class index_node
{
    std::vector<int> feature_vector;
    std::set<object_ptr> objects;
};
```

In MVP, the objects attribute would have only *one* object. This is a generalization of the form

```
class traditional_index_node
{
    int id;
    object_ptr object;
};
```

Which is what you find in typical content systems.

5.3 Have a Simple Selection Function

Given our index_node object described above, we can have a selection function that looks like this.

```
object_ptr select(index_node& n)
{
    return n.objects[0];
}
```

This would obviously be a equivalent to this if we had the traditional_index_node

```
object_ptr select(traditional_index_node& n)
{
    return n.object;
}
```

5.4 Store Statistics to /dev/null

We have a place in the system to count stats like clicks and impressions.

```
void store_stat(time_t, string, int);
```

Where the implementation in the first go looks like this.

```
void store_stat(time_t timestamp, string key, int value)
{
//do nothing
}
```

In MVP we would not even write a function to retrieve stats.

6 Summary

We have an opportunity to innovate how content is managed and displayed. I proposed here a generalization of how most content systems store and retrieve content.

By having a place in our architecture to select indexes based on feature vectors, where the index nodes point to a set of objects, and we store stats in a time series database, we can create new ways to manage content and have a system that adapts to changes in customers and demand.

The best part, this system can be built just-in-time using a small set of architectural decisions early on.