Sam Zhang Follow
Sam is a software engineer and a Data Science for Social Good Fellow. https://sam.zhang.fyi.
Jan 22 · 7 min read

How OpenCounter designed a use code search engine for local governments

Consider the local businesses in your neighborhood: coffee shops, yoga studios, salons, gyms.

When entrepreneurs come to City Hall to open a new business, they will immediately learn that "yoga studio" is not an official term. City staff can usually make sense of their descriptions, and translate them into the official terminology of the zoning code, but this process is based on years of experience and deep familiarity with the regulations. Automating the process in software reveals how complicated the translations really are, and leads to challenging computer science problems related to semantics and search.

A quick primer on zoning and land use

Every city employs a unique set of *land use*s, which are descriptions of business activity that help to determine where businesses are allowed to operate. We refer to the classifications as *land use codes*, or just *use codes*.

Although there are some similarities that most bodies of land use codes share, there is no standard or requirement for their structure. Across jurisdictions, there will be different numbers of land uses, different names for similar uses, different categorizations for the uses, and uses will be specified at different levels of detail.

"Yoga," for instance, is usually grouped under "Commercial Services" and can be called "Personal Instruction," "Small Indoor Recreation," or "Group Fitness Facility." In contrast, gyms can be grouped under either "Commercial, Services" or even "Recreation, Public Assembly," with the actual use as "Recreation."

Cafés are often grouped under "Commercial, Eating and Drinking Establishment," but depending on the local code, can be further

categorized as a “Limited Service Eating and Drinking Establishment” (meaning that no table service is provided).

Salons are often grouped with tailors, laundry shops, and sometimes even tattoo parlors in a land use called “Personal Services”, under the umbrella of “Commercial, Services.”

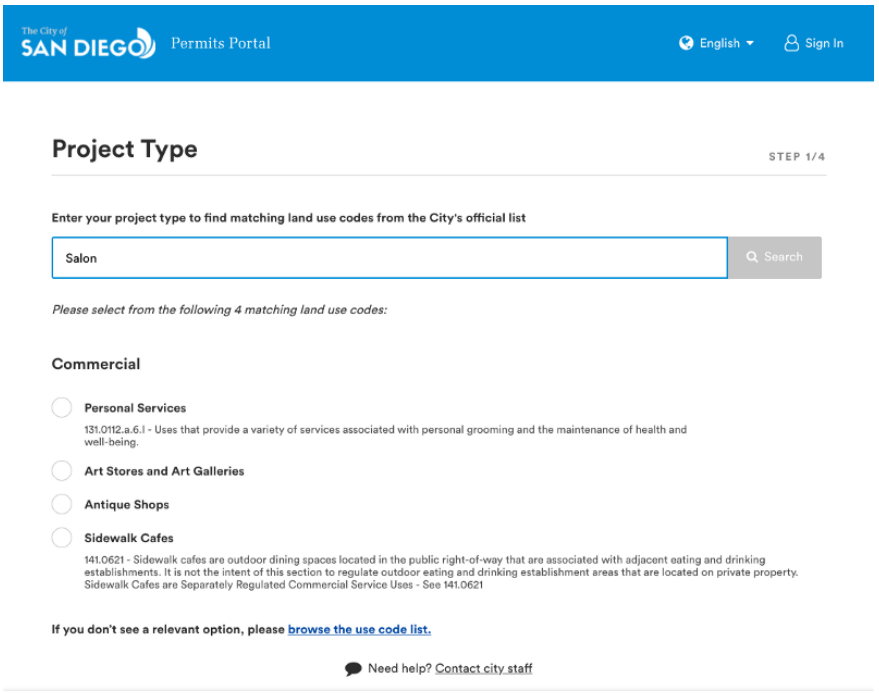
These technical terms are not always intuitive for people new to the process, but the choice of land use code can have a significant impact on where the business is allowed, meaning that choosing the right option is a crucial step in the overall permitting process.

. . .

At OpenCounter we’re always trying to make interacting with local government as seamless as possible. This is why when someone comes to OpenCounter to check their zoning or research permits and fees for their business, we’re able to replicate the conversations that happen at the counter by matching common business terms to land uses defined by the City. If cities have definitions for uses, those are available to users as well.

OpenCounter already has the ability to show the user a map of where their use code is allowed. Now the process of finding a use code is simplified too.

Over the last six months, over 90% of users found a use code through the search without resorting to browsing the land use table.



A screenshot of the use code search page of https://business.sandiego.gov/

How do we do this? We use a technique at the intersection of natural language processing and machine learning called word embeddings to generate a graph representation of all of the use codes across all of the jurisdictions in OpenCounter, and use results from other jurisdictions to inform any particular search. As OpenCounter grows and is able to compile more uses and associate more words, this algorithm becomes more accurate.

For example, when a user searches for “salon”, we find “Personal Services” because “salon” has a short distance with words like “barber”, “spa”, “beauty”, and “nail”, which our system has already associated with “Personal Services”. For more details on the graph nature of the search, see the technical explanation below.

Technical explanation

The motivation for constructing a use code graph on top of word embeddings is threefold:

1. We are unable to make a single search engine that takes a query, and maps onto a global taxonomy of uses, since jurisdictions have slightly (and sometimes very) different sets of use codes.

2. We don't want individual search engines for each jurisdiction, because we want to exploit the similarities between the use code sets to improve the quality of the search.

3. We want to minimize the amount of human annotation that needs to go into the system. Therefore we want to fuzzy match on words that OpenCounter hasn't needed to explicitly associate with the system. Word embeddings allow this. For example, we want "pilates" to have a close association with any uses we know to be related to "yoga".

Embedding use codes

A word embedding is an embedding from a space where each word is one dimension to a continuous vector space. We use an implementation of word embeddings developed at Stanford called GloVe [1].

We use a pre-trained model of the top 400,000 thousand words from English Wikipedia, projected onto a hypersphere.

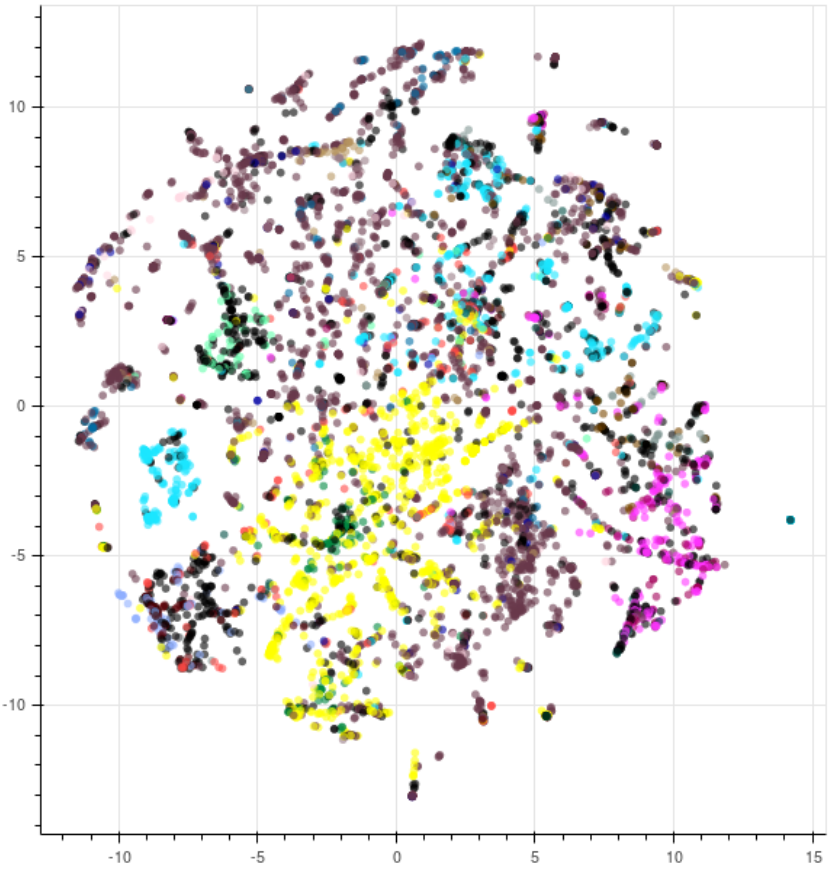
At the beginning, we treat the use code as a bag of words $[w_0, w_1, \dots, w_n]$, which consists of the category, subcategory, name, and any additional keywords associated with a given use code. We embed each word individually from this bag of words to form a bag of vectors $[p_0, p_1, \dots, p_n]$. Then we combine these vectors into a single vector $\vec{v}$ using tf-idf.

Use codes are allowed to have multiple vector representations, since a use like "Commercial Administrative" can be the correct result for "law firm" as well as "advertising" or "software consulting"—words that aren't necessarily close in the vector space. This is accomplished by giving each use code multiple "keyword sets" that append onto the name of the use itself. For the sake of simplicity, the rest of this article will treat use codes as the fundamental unit, rather than keyword sets.

Each use code is then represented as a tuple of $(j, \vec{v}), j \in J$, where $\vec{v}$ is the vector representation of the use code, and J is the set of all jurisdictions in OpenCounter, which are the clients in OpenCounter such as San Diego and Salt Lake City. Thus a use may share the same position in vector space with a use in a different jurisdiction, but is assumed to be unique within its own jurisdiction.

Each jurisdiction has its own set of use codes U_j, which partition the set of all use codes in OpenCounter U.

Here is a visual representation of U in two dimensions, created using t-sne [2] and bokeh [3]:



t-sne representation of the vector space of uses

Each use code is a point, and each color corresponds to a top-level use category, such as “commercial”, “industrial”, or “residential”.

In the interactive version of the above plot, the user can move their cursor across the points to watch how the use codes change: https://sam.zhang.fyi/html/use-code-tsne.html. (Warning: 13MB html file)

Exploiting the use code graph

We use cosine distance as a distance metric D between use codes. The user’s query vector will be denoted $\vec{q}$, and the jurisdiction of the user t.

We find that searching for the closest match within the target jurisdiction from the query—minimizing $\{ D(\vec{q}, u) \mid u \in U_t \}$—performs poorly. The main reason this happens is because often a particular jurisdiction *simply doesn't have detailed use codes*. It is much harder to match the query "office" to "Commercial > General Administrative" than it is to "Commercial > General Administrative > Office".

The quality increases when we rely on the underlying structure of the graph to allow use codes to influence each other across jurisdictions. The naive approach of minimizing $\{ D(\vec{q}, u) \mid u \in U_t \}$ can be viewed as a minimization of the path $\sum p_i$, where $p_n \in U_t$ and $p_0 = \vec{q}$ (this is the great circle distance across the hypersphere between $\vec{q}$ and its nearest point in U_t). Instead, to make use of the prior structure within the graph, we minimize the length of each step $argmin \{ D(p_i, p_{i+1} \mid u \in U \wedge u \notin \{ p_0, p_1, \dots, p_i \}) \}$. This opens up a potentially large search tree, but we find it sufficient to perform only one step.

In other words, for each $u \in U$, we save the precomputed use $s(u, j) = argmin \{ D(u, u_j) \mid u_j \in U_j, j \in J \}$. Then $n=2$, and $p_2 = s(p_1, t)$. This creates a precomputed graph G of size $|U| \times |J-1|$, where for every given use code, we find the closest use code in every other jurisdiction.

A hypothetical search

Suppose there is a city with a use code called "Commercial > Administrative", and that use code is the correct result for "office".

When the user types in "office", we first find the top N (say, 20) matches across the entire system, across all jurisdictions. It would likely be a group of use codes all similarly named to "Commercial > General Administrative > Office".

Then for each of those candidates for p_1, we look up $s(p_1, t)$ within G, and rank the results using a linear combination of $D(\vec{q}, p_1)$ and $D(p_1, p_2)$.

This is essentially a form of query expansion, where we rely on the prior knowledge of all of the use codes in OpenCounter to expand our query from $\vec{q}$ to p_1. This is why adding as OpenCounter grows, and more use codes are added to the system, the quality of the search will continue to improve.

Productionization notes

This system was developed in Python with the gensim package [4], but to avoid maintaining a microservice in a separate programming language than our existing Rails application, we migrated the word vectors into Postgres to use with ActiveRecord. We store the data using the Postgres "cube" extension [5], which provides us with a high-dimensional cube data structure as well as performant distance functions.

Footnotes

[1] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.

[2] Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of Machine Learning Research* 9.Nov (2008): 2579–2605.

[3] https://bokeh.pydata.org/en/latest/

[4] https://radimrehurek.com/gensim/

[5] https://www.postgresql.org/docs/9.5/static/cube.html