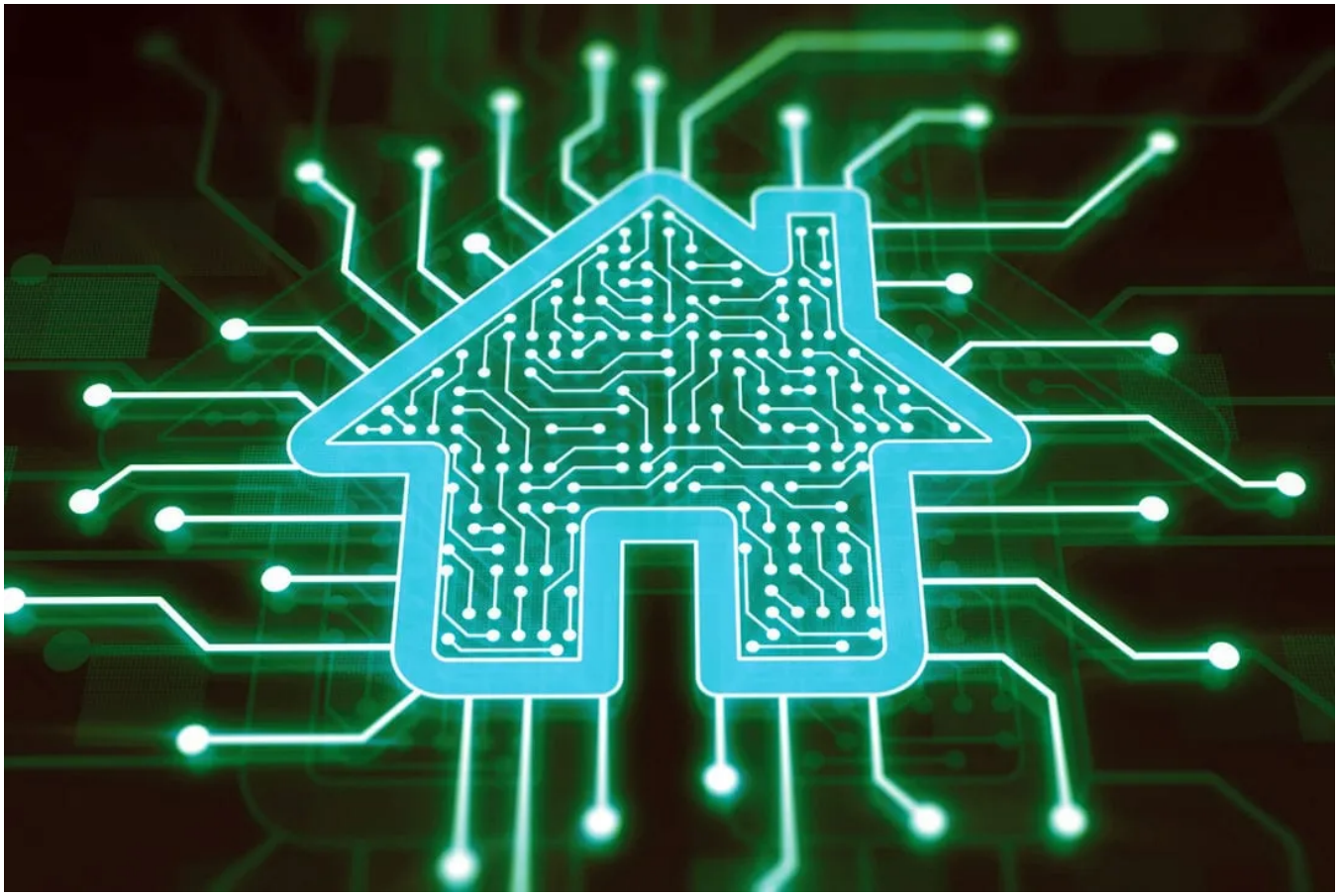
A Street Group Address Matching Algorithm

Juan Saiz-Lomas · Follow
Published in StreetGroup
6 min read · Sep 1, 2022



Credit — StartupEssence

Address matching — the process of identifying pairs of address records referring to the same spatial footprint — is increasingly required for enriching data quality in a wide range of real-world applications.

Street Group is one of the fastest-growing PropTech companies in the UK, and is dedicated to improving the experience of agents and their clients whilst they navigate the process of moving home. Our broad range of innovative software products relies on the quality and consistency of our UK property database. The process of matching the original raw-string addresses from our various database ingestion sources to their respective standard Royal Mail postal address Unique Property Reference Number (UDPRN) plays a fundamental role in ensuring the reliability of our data. However, the linkage of these raw-address records can present a challenge to unlocking the full potential of our integrated (spacial) data sources.

For this purpose, we designed a state-of-the-art address matching model to link the input raw-addresses to their corresponding Royal Mail UDPRNs.

The Machine Learning / NLP solution

With an input raw-address and its postcode, we can look up the Royal Mail database to obtain the corresponding UDPRNs and addresses for that postcode. The goal is to develop an NLP solution that will score the similarity between address elements (see table below). Using a simple rule-based model, it is possible to obtain certain address matches. However, as source raw-addresses become more disorganised and unformatted, this approach won't always work — e.g. " *10A 3 FLOOR FULHAM BROADWAY "* could confuse a rule-based model to select *"flat 10 3 fulham broadway"* as opposed to *"third floor flat 10a fulham broadway"*. This indicates that there is significant room for improvement using some more advance NLP /ML techniques.

Input raw-address	Input postcode	UDPRN (encoded)	RM address	Postcode
10A 3 FLOOR FULHAM BROADWAY	SW6 1AA	eyJ1ZHBybiI6ICIyNzY3NzYxMSJ9	11 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ0NyJ9	flat 2 7 fulham broadway	SW6 1AA
		eyJ1ZHBybiI6ICIyMzgzNTYyNiJ9	sia hair & beauty 8 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ0OCJ9	flat 3 7 fulham broadway	SW6 1AA
		eyJ1ZHBybiI6ICIyMzgzNTYzNiJ9	william hill bookmakers 12-14 fulham b	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ1MyJ9	flat 8 7 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NjAzNTQyNCJ9	second floor flat 10a fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ1MCJ9	flat 5 7 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NjAzNTQyNiJ9	third floor flat 10a fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1MjU1NjY0MCJ9	flat b 12-14 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ1MSJ9	flat 6 7 fulham broadway	SW6 1AA
		eyJ1ZHBybiI6ICIyMzgzNTYyNCJ9	whole foods 3 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1NTEyODQ1NCJ9	flat 9 7 fulham broadway	SW6 1AA
		eyJ1bXBybiI6ICI1MjU1NjYzOSJ9	flat a 12-14 fulham broadway	SW6 1AA
		eyJ1ZHBybiI6ICI1NDAwOTA0MyJ9	flat 10 3 fulham broadway	SW6 1AA

Our current model uses multiple different rules to assess whether a pair of addresses is a match. This model leaves a big proportion of our dataset un-matched due to the above mentioned complications when the input raw-address has a different format. However, the input addresses that the current model manages to find a UDPRN for can be used as a dataset for training a machine learning model.

Provided we know the matching UDPRN for several million properties in the UK, we can generate a highly skewed training dataset for binary classification, such as:

Input raw-address	RM address	Label
10A 3 FLOOR FULHAM BROADWAY	11 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 2 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	sia hair & beauty 8 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 3 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	william hill bookmakers 12-14 fulham b	0
10A 3 FLOOR FULHAM BROADWAY	flat 8 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	second floor flat 10a fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 5 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	third floor flat 10a fulham broadway	1
10A 3 FLOOR FULHAM BROADWAY	flat b 12-14 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 6 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	whole foods 3 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 9 7 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat a 12-14 fulham broadway	0
10A 3 FLOOR FULHAM BROADWAY	flat 10 3 fulham broadway	0

In order to generate the features to be fed into our ML model, we followed some of the preprocessing steps from *Y. Lin et al. (2020)*:

1. We parsed all address strings with the well established pre-trained Conditional Random Field (CRF) address parser library *libpostal*.

```
>> parse_address('10a 3 floor FULHAM BROADWAY', language='EN',
country='GB')

[('10a', 'house_number'), ('3 floor', 'level'), ('fulham broadway',
'road')]
```

There are additional address-elements supported by *libpostal* and these can be provided as padding generating an address-element vector of the form:

```
vector1 = [nan, nan, nan, '10a', nan, nan, '3 floor', nan, nan, nan,
nan, 'fulham broadway', nan, nan, nan, nan, nan, nan, nan, nan]
```

Similarly for the corresponding Royal Mail address pair

```
>> parse_address('third floor flat 10a fulham broadway',
language='EN', country='GB')

[('10a', 'house_number'), ('third floor flat', 'level'), ('fulham
broadway', 'road')]
```

and corresponding address vector

```
vector2 = [nan, nan, nan, '10a', nan, nan, 'third floor flat', nan,
nan, nan, nan, 'fulham broadway', nan, nan, nan, nan, nan, nan, nan,
nan]
```

2. Compute the Jaro-Winkler similarity in between address-element vector-components generating a comparison vector.

```
comp_vector = [0. , 0. , 0. , 1. , 0. , 0. , 0.3, 0. , 0. , 0. , 0.
, 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ]
```

3. Train FastText embeddings on all Royal Mail addresses using the address-elements given by *libpostal* as tokens. Once trained, calculate the cosine similarity between the embedding representation of *vector1* and *vector2* and add the result to the *comp_vector*.

4. Calculate the Jaccard similarity between the input strings.

5. Add additional comparative features between input strings such as number of non-null address-elements in *vector1* and *vector2* respectively, and number of characters in each input address strings.

The resulting comparative vector for each address pair can be formatted into a pandas DataFrame as illustrated below. Often the cosine similarity (*cossim*) and Jaccard similarities (*jaccard_similarity*) will give a strong indication of what the correct address match is but this won't always be the case.

	house	category	near	house_number	road	unit	level	staircase	...	world_region	jaccard_similarities	count_notNan_1	count_notNan_2	sum_notNan_1	sum_notNan_2	cossim	labels
0	0.0	0.0	0.0	1.00	1.0	0.0	0.0	0.0	...	0.0	1.000000	2.0	2.0	18.0	18.0	1.000000	1
1	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.666667	2.0	2.0	18.0	19.0	0.999807	0
2	0.0	0.0	0.0	0.61	1.0	0.0	0.0	0.0	...	0.0	0.785714	2.0	2.0	18.0	19.0	0.999788	0
3	0.0	0.0	0.0	0.61	1.0	0.0	0.0	0.0	...	0.0	0.785714	2.0	2.0	18.0	19.0	0.999822	0
4	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.714286	2.0	2.0	18.0	19.0	0.999754	0
5	0.0	0.0	0.0	0.61	1.0	0.0	0.0	0.0	...	0.0	0.846154	2.0	2.0	18.0	19.0	0.999774	0
6	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.714286	2.0	2.0	18.0	19.0	0.999759	0
7	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.666667	2.0	2.0	18.0	19.0	0.999781	0
8	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.785714	2.0	2.0	18.0	19.0	0.999746	0
9	0.0	0.0	0.0	0.00	1.0	0.0	0.0	0.0	...	0.0	0.714286	2.0	2.0	18.0	19.0	0.999776	0

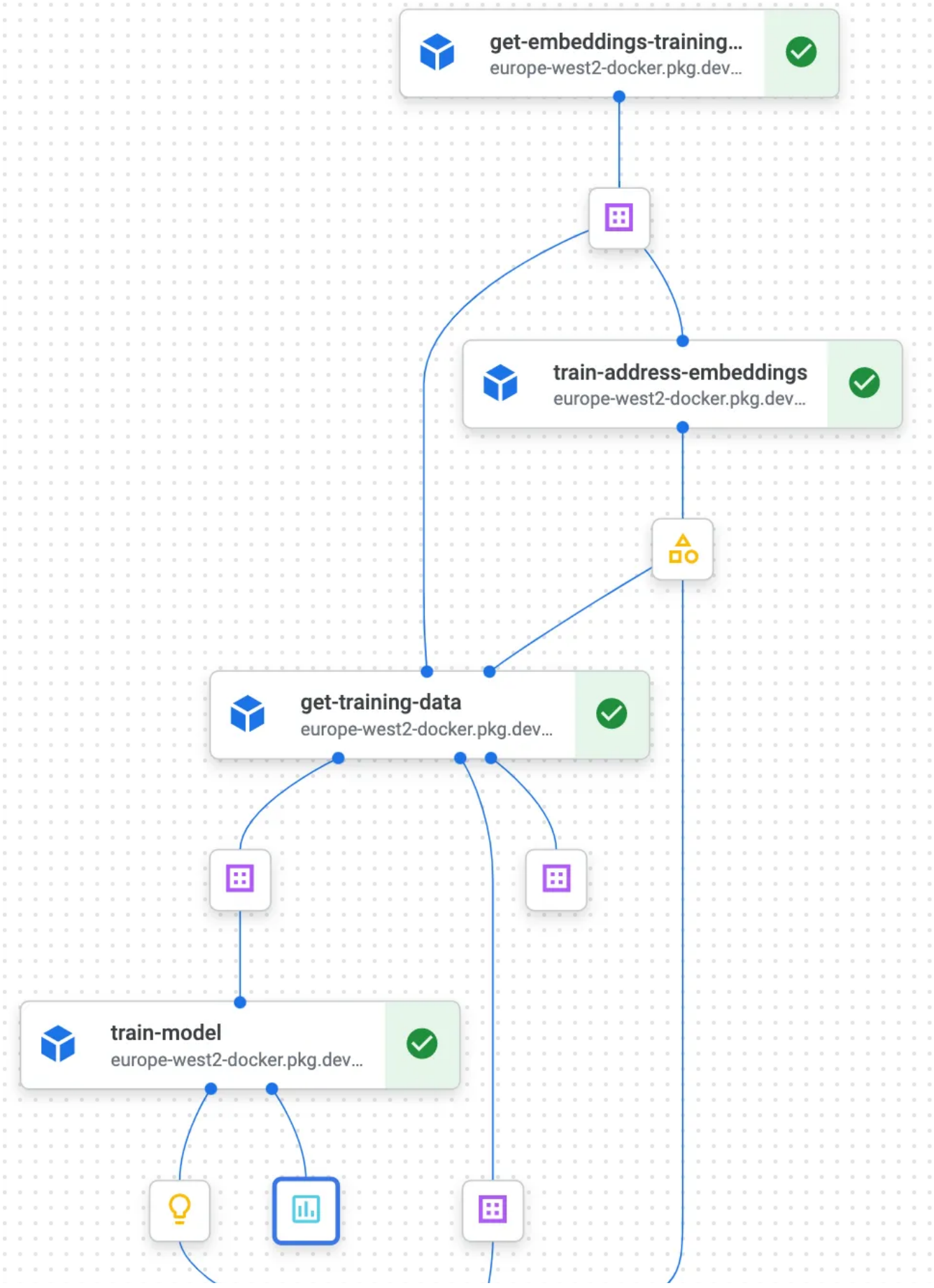
Finally, we trained an XGBoost binary classifier to predict whether a pair is a match based on the generated comparative features. Class weights needed to be passed to the constructor of the classifier, since there were about 95% of non-matching addresses (label 0) vs 5% of matches (label 1). As such, the contribution of the positive class to the loss function was scaled up to account for the class imbalance.

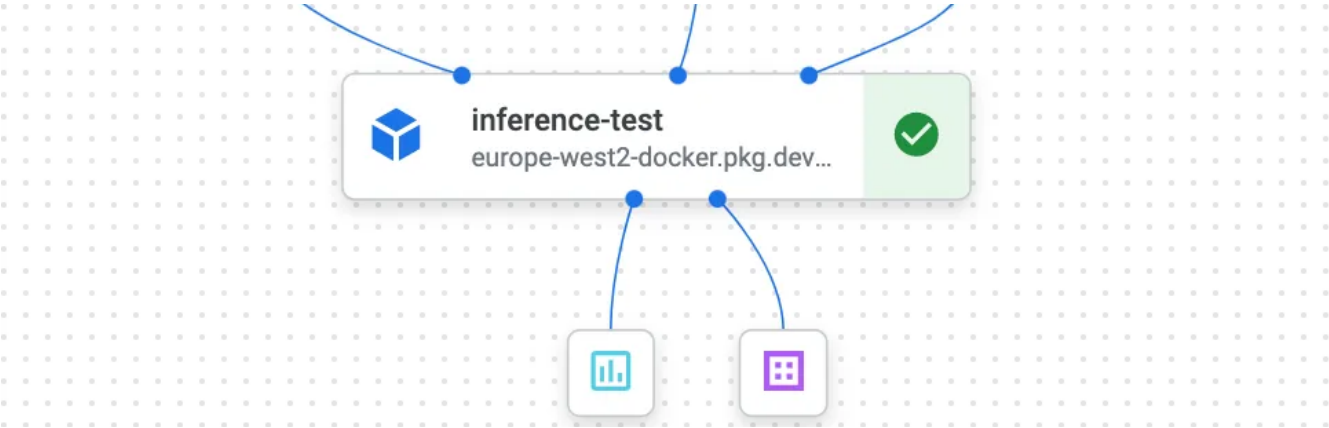
The model achieved a 99.1% precision on the known UDPRN dataset, the remaining

0.9% constituting errors made by the rule-based model that generated the labels. In addition, the model allowed us to address-match an additional 80% of the addresses without UDPRN, of which approximately 60% had a high confidence score (*i.e.* high XGBoost probability).

Deployment

The training of the embeddings, generation of the comparative features and training of the XGBoost classifier were deployed into a Kubeflow pipeline. The high scalability and exceptional metadata storage made Kubeflow one of our preferred deployment options. Kubeflow allowed us to customise the CPU and RAM needed at each step-component of the pipeline, which ultimately resulted in a cost optimisation.


get-embeddings-training...
europe-west2-docker.pkg.dev...
train-address-embeddings
europe-west2-docker.pkg.dev...
get-training-data
europe-west2-docker.pkg.dev...
train-model
europe-west2-docker.pkg.dev...



In addition to the Kubeflow pipeline, we also needed to have this model accessible via an API endpoint. The endpoint would fetch the necessary metadata objects from the pipeline (*i.e.* the trained model, the trained embeddings and the dictionary of addresses and postcodes) and, when given a raw-address and a postcode, returned an encoded UDPRN and the confidence score for the match — we chose FastAPI to serve the model. The deployment of this endpoint proved challenging due to the long loading times of the model within the created Docker image; the dictionary of addresses and the FastText embeddings are fairly large and take several minutes to load into memory, resulting in complications when using GCP cloud run. Finally, the endpoint was deployed using standalone Kubernetes with the model objects still being fetched from the Kubeflow pipeline.

Conclusion

This project was very instructive both from the perspective of developing the NLP model and its deployment. Other NLP solutions such as SiameseLSTM or simple cosine similarity models were also explored but the combination of CRFs and string similarity metrics with an XGBoost model proved to be the highest scoring in our data.

References

- Y. Lin *et al* (2020). *A deep learning architecture for semantic address matching*, International Journal of Geographical Information Science, **34:3, 559–576**, DOI: 10.1080/13658816.2019.1681431

- S. Comber and D. Arribas-Bel (2019). *Machine learning innovations in address matching: A practical comparison of word2vec and CRFsMachine learning innovations in address matching: A practical comparison of word2vec and CRFs*, Transactions in GIS, **23:334–348**, DOI: 10.1111/tgis.12522
- Rui Santos, Patricia Murrieta-Flores & Bruno Martins (2018). *Learning to combine multiple string similarity metrics for effective toponym matching*, International Journal of Digital Earth, **11:9, 913–938**, DOI: 10.1080/17538947.2017.1371253

NLP | Machine Learning | Geocoding | Data Science | AI

Follow

Written by Juan Saiz-Lomas

5 Followers · Writer for StreetGroup

Machine Learning Engineer @ Street Group

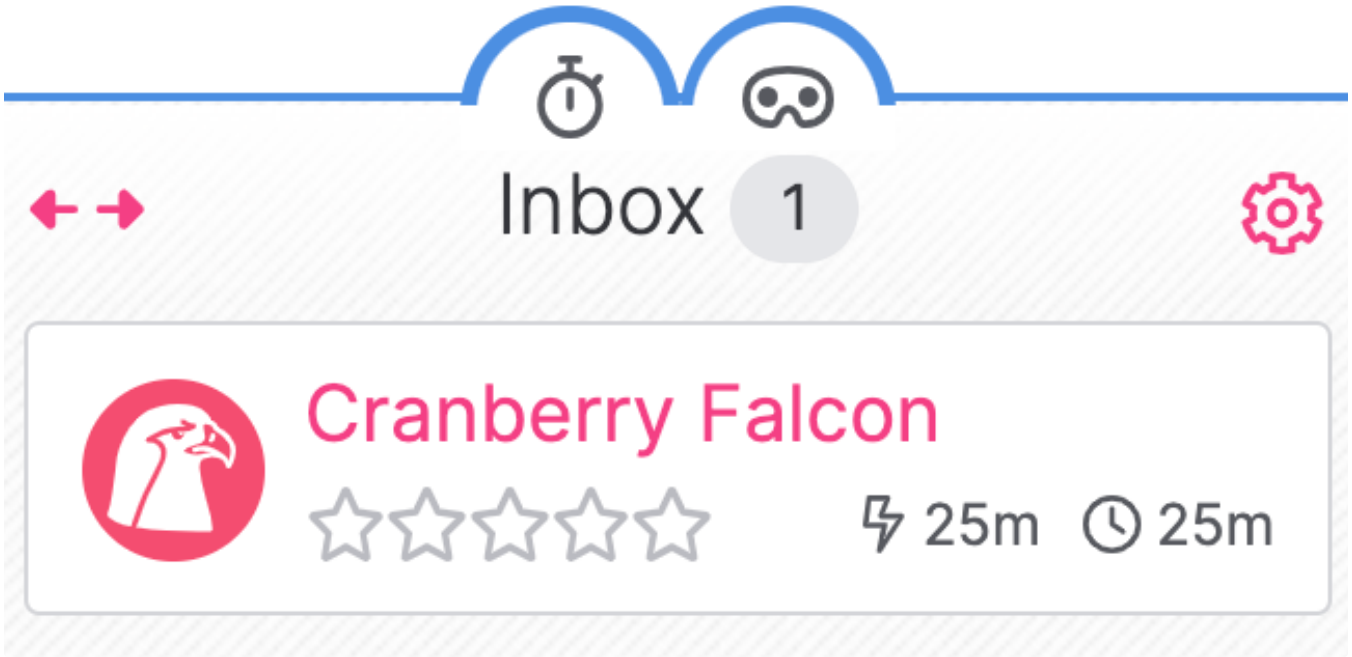
More from Juan Saiz-Lomas and StreetGroup

Dom Ioanna in StreetGroup

Life as a Software Engineer at Street Group

Dom, you've been at Street Group for nearly 2 years now, how's the journey been so far?

10 min read · Feb 26, 2024



Laura Whitworth in StreetGroup

Interviewing at Street Group: what to expect

Everything you need to know about interviewing at Street Group!

6 min read · Nov 26, 2021

7

Dom Ioanna in StreetGroup

StreetCon 2024

It's here! It's conference day!

5 min read · Feb 22, 2024

Luke Almond in StreetGroup

Joining Street Group as a Junior Software Engineer

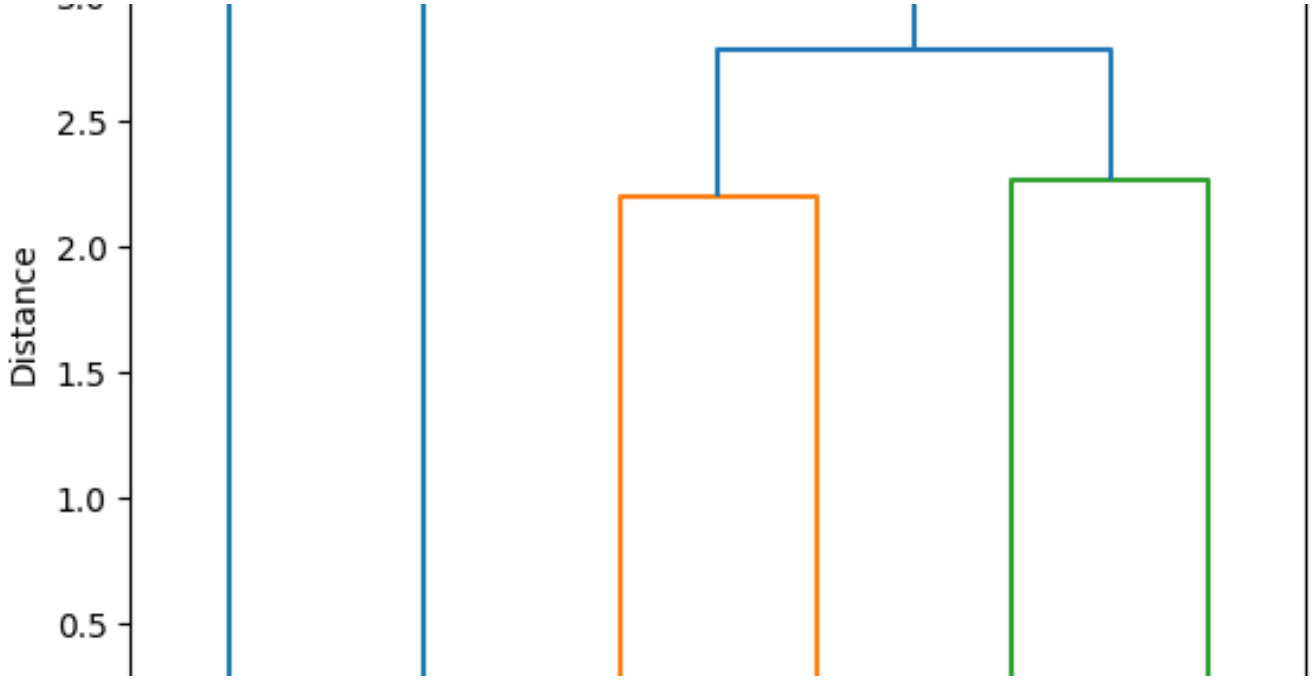
Luke, how has your journey at Street Group been so far?

3 min read · Jan 29, 2024

See all from Juan Saiz-Lomas

See all from StreetGroup

Recommended from Medium



NANDINI VERMA in GoPenAI

Comprehensive Overview of Hierarchical Clustering: Agglomerative and Divisive Approaches...

Hierarchical clustering is a method of cluster analysis that builds a hierarchy of clusters. This technique can be visualized as a...

3 min read · Dec 14, 2023

8

ihsaan.patel

Are these two people the same?

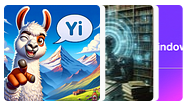
Solving entity resolution with LLMs

3 min read · Nov 15, 2023

Lists

Predictive Modeling w/ Python

20 stories · 1030 saves



Natural Language Processing

1315 stories · 806 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 348 saves

Practical Guides to Machine Learning

10 stories · 1232 saves

Nancy Oglesby in Fiction Shorts

A Missing Inventory Mystery

A perplexing case

· 2 min read · Feb 22, 2024

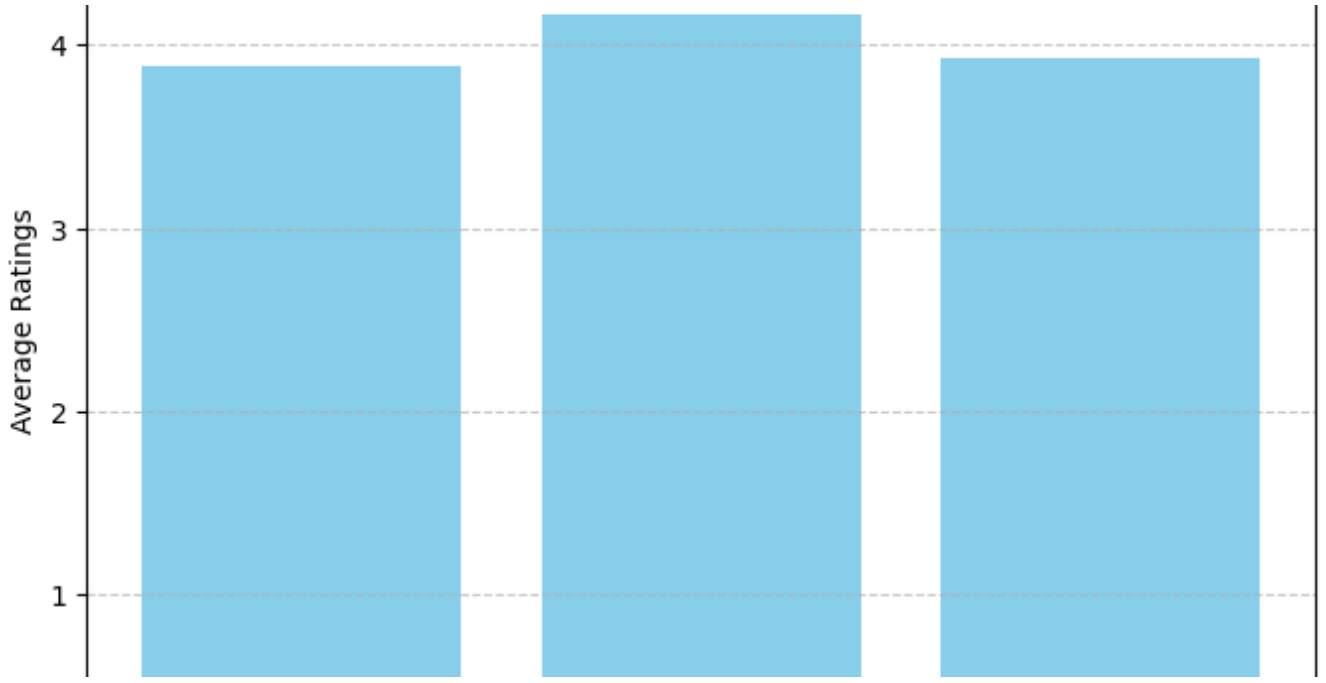
Tomonori Masui in Towards Data Science

Entity Resolution: Identifying Real-World Entities in Noisy Data

Fundamental theories and Python implementations

19 min read · Sep 20, 2023

333 1

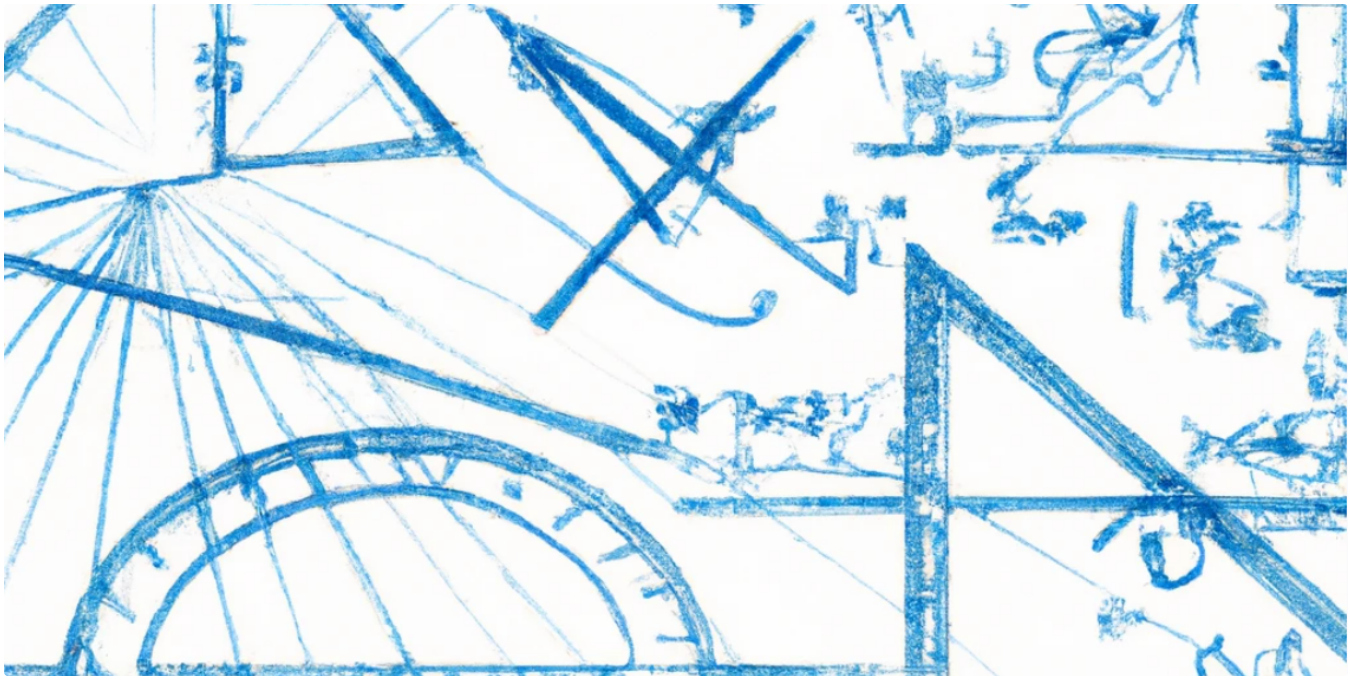


James van Doorn in INST414: Data Science Techniques

Exploring Similarity Queries in Web-Based Data

Non-obvious Insight

3 min read · Oct 27, 2023



DataStax

How to Implement Cosine Similarity in Python

By Phil Miesle

4 min read · Nov 30, 2023

13

See more recommendations