


PlaceRank

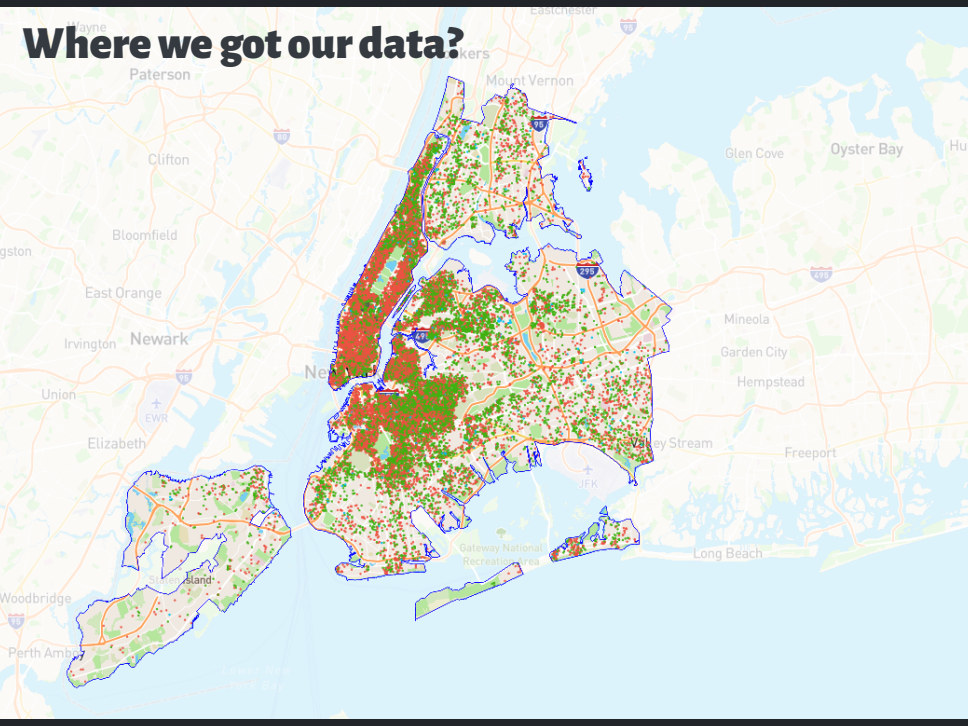
full-text search engine on Airbnb data

Giulio Corradini, Francesco Mecatti, Antonio Stano

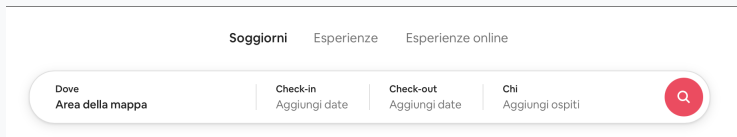
Università di Modena e Reggio Emilia

 github.com/mc-cat-tty/PlaceRank

Where we got our data?



Searching on Airbnb

The image shows the Airbnb search bar interface. At the top, there are three tabs: "Soggiorni" (selected), "Esperienze", and "Esperienze online". Below the tabs is a search bar with four sections: "Dove" with the placeholder "Area della mappa", "Check-in" with the placeholder "Aggiungi date", "Check-out" with the placeholder "Aggiungi date", and "Chi" with the placeholder "Aggiungi ospiti". A red search button with a magnifying glass icon is located on the right side of the search bar.

Soggiorni Esperienze Esperienze online

Dove
Area della mappa

Check-in
Aggiungi date

Check-out
Aggiungi date

Chi
Aggiungi ospiti

Q

Airbnb **search** doesn't provide **free text search** capabilities.

- Users can't just "see what's there".
- **Listings** can only be filtered by categories, such as "by the sea", "castles", "for creatives", but they are way too vague.
- With **no filters** applied, the map shows results that are only a **fraction of potentially thousands available**. Mostly non-relevant to the selected zone.
- **Reviews** content is **not taken** into account. Star ratings only.

Exploring Inside Airbnb

InsideAirbnb is a non-profit that scrapes listings and reviews from Airbnb

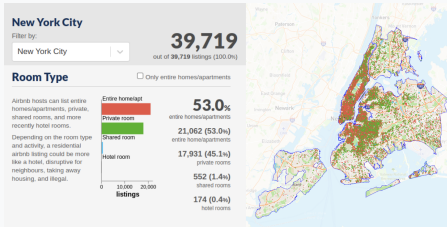


Figure: **insideairbnb**

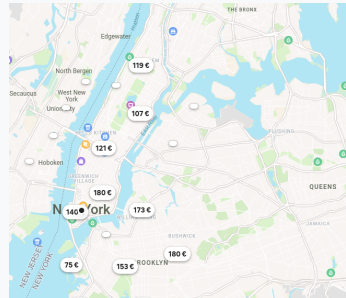


Figure: **airbnb**

Benchmarking

We developed a **custom benchmark** set from **Cambridge listings**.

- **10** different **UINs**
- **Expected relevant** results are returned in **ranking order**
- Expected sentiment specified by the user, related to the query.

Precision and **recall** were used as metrics, as they don't require human interventions in computation (while the alternative, DCG, does).

Extending the space vector model

To enhance the precision we considered:

- using **word embeddings** and a custom tree index that clusters synonyms together, assuming "near" embeddings represents synonyms, and get query expansion for free;
- or apply **sentiment analysis** and utilize the reviews dataset, which Airbnb doesn't exploit except for star ratings.

Assumptions

We presume that the user is conducting searches to find their **ideal** apartment through the system.

Characteristics are best expressed with **natural language**. Every part of speech is indexed.

Example query: "sunny flat" should yield flats with big windows towards the sun.

Listings may not reflect the reality when the host is not transparent. Reviews content is exploited to improve retrieval.

Short queries do not broadly cover all the range of **emotions**, as they can be mostly **neutral** very often. The user **may specify** the sentiment **vector** by adding **tags**.

Support data structures

We used the following **indexing structures**:

- **Inverted index.** Managed by Whoosh. Listing name, description and neighborhood overview are stored there.
- **Forward index.** Indexed by listing ID, and postings made up of (sentiment vector, date, review ID) tuples.

The forward index is stored **separately** making our system **fast** on reviews **insertion and deletion**; it also allows us to choose the **averaging function** arbitrarily.

Query language

The system provides a **free text search field**, where queries in a keyword-based fashion can be performed. Additional **filtering** based on **room type** is achieved with a **check box vector**.

The system, if a **sentiment-aware IR model** is enabled, will **compare the sentiment** of the query that the user may **specify**.

There are **27 available sentiment labels**, and listing similarity is computed with **cosine similarity**. Listings are ranked using BM25F.

System architecture

We customized some **Whoosh** core elements:

- **MultifieldUnionPlugin** extends `MultifieldPlugin`, generates an AST with OR as a root.
- **SentimentWeightingModel** enables sentiment ranking.

Why we put the terms in OR?

Let's make some **considerations**.

Whoosh **by default** combine every term in a query with the **AND** operator, but this **impacts** both the **retrieval** and **ranking** function.

With **OR terms**, however, the work is entirely **delegated** to the **ranking function**.

This is achieved by creating a custom **"Proxy" Plugin**, that edits MultifieldParser AST.

Boolean queries are still possible. AND and NOT operators are kept inside the AST supplied to the parser.

Increasing recall with query expansion

We chose a **global, query-independent**, expansion strategy. Two sub-strategies have been implemented. Both work by:

1. generating a set of **alternative terms** for each word
2. **similarity** against the original query (BERT embeddings)
3. **filtering out** terms that **drift** the original semantic meaning

Candidates generation:

- **WordNet-based**: candidates are WordNet synonyms.
- **BERT-based**: MLM is exploited with **fill-in the gap** task

Eg: modern shared room near Harvard becomes

- WordNet-based: {innovative, advanced} {shared out, divided} {elbow room} {close, draw near} {John Harvard, Harvard University}
- BERT-based: {living} {} {house, residence} {in, of}

Sentiment Ranking

Cosine similarity computes the similarity score between **user's** binary **sentiment vector** and **listing's** real **sentiment vector**.

To allow sentiment negation to the user, a third state is introduced: -1. The unary NOT operator flips the **sign** of sentiment's **weight**, making it **negative**.

Eg: disappointment disapproval NOT disgust becomes something like (0 ... 0 1 1 0 ... -1)

More details about sentiment classification later in the slideshow.

Host Redemption Model

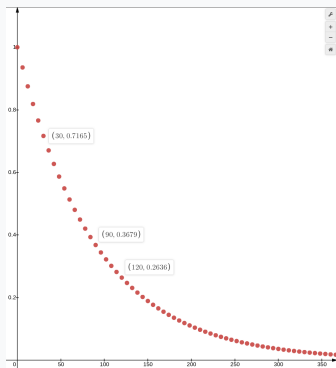


Figure: **Weight's exponential decay vs days from last review**

$\tau = 90$

Given a set of sentiment vectors, their weights are progressively **damped** through an **exponential decay**.

This model, named **host redemption model**, takes into account that AirBnB hosts can change their behaviour over time: older reviews are **less influential** than newer ones.

Let \bar{d} be the last review of a listing; R the set of its reviews and d_i the day of the i -th review:

$$w_i = e^{\frac{-(\bar{d}-d_i)}{\tau}} \quad ds(R) = \sum_{i=1}^{|R|} w_i s(r_i)$$

Scores combination function

The first approach combined **textual similarity** and **sentiment similarity** with a naive **product**.

A problem arose during the testing: highly matching reviews made the **associated listing** go way **up** in the **ranking**.

A more realistic approach is **weighting** the final score with a **coefficient** proportional to the number of reviews for the listing: more reviews mean **more trust** towards the host; they are more likely to represent the truth.

Let s be a score value, l_i be a listing and R_i be the reviews of that listing.

$$s'(l_i) = s(l_i)len(R_i)$$

In our case $len(R_i)$ is clamped between 0 and 10.

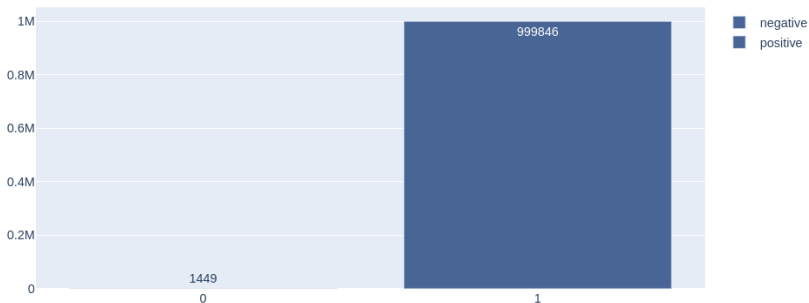
Finetuning BERT for sentiment analysis

critical issues encountered with:

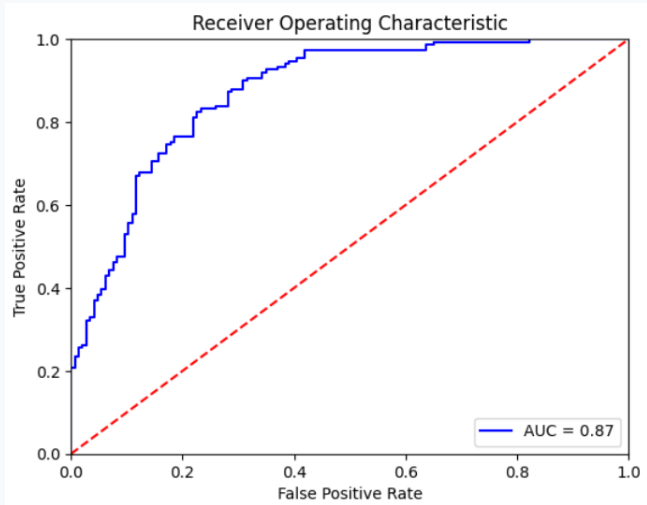
- Dataset **EDA**
- Sentiment **labels**
- Baseline: **TF-IDF** + Naive **Bayes Classifier**
- Fine-tuning **BERT** for **binary classification**
- **Evaluation** on Validation Set
- **Predictions** on **Test Set**
- **classification** using **vanilla BERT**

NYC reviews dataset EDA and labels criteria

Dataset distribution by target



Fine-tune and training loop over validation set



AUC: 0.8723

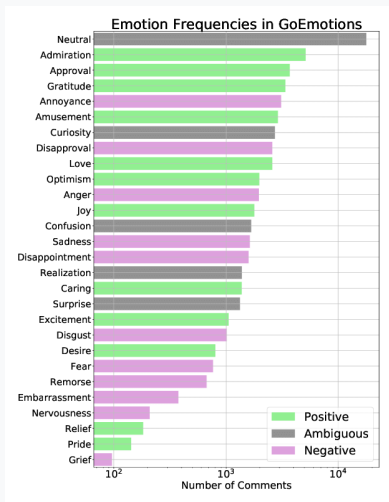
Accuracy: 79.32%

Predictions: finetuned vs vanilla

The **test dataset** is made by combining all the **negative reviews** with a sample of **105 non-negative reviews** (always based on our sentiment labels criteria). The goal of evaluating the model over the test set is by **predicting** how many reviews it **classifies** as **non-negative**.

- **predictions with BERT finetuned** on all the train set
 - **threshold = 0.9177**
 - Number of reviews predicted **non-negative: 108**
- **predictions with BERT vanilla** ('bert-base-uncased' model)
 - **threshold = 0.5111**
 - Number of reviews predicted **non-negative: 109**

GoEmotions: A Dataset of Fine-Grained Emotions



GoEmotions label frequencies

Understanding emotions expressed in language has a wide range of applications, from building empathetic chatbots to detecting harmful online behavior.

GoEmotions is a corpus of 58k carefully curated comments extracted from Reddit, with human annotations to **27 emotion categories** or **Neutral**.

Multi-label text classification

We use a **pytorch** implementation of **GoEmotions** with HuggingFace **transformers**.

The model is '**monologg/bert-base-cased-goemotions-original**', which is made by fine-tuning a BERT-base model following the instructions of this google-research paper: "**GoEmotions: A Dataset of Fine-Grained Emotions**".

The model allows us to **classify** any given input texts and output the relative scores of each label, in order to perform **sentiment analysis** tasks. In our specific case, we apply the classification to the **reviews dataset**.

Future **IMPROVEMENTS** (clustering trees, LLM embeddings...)

And **now**... a live **demo**!