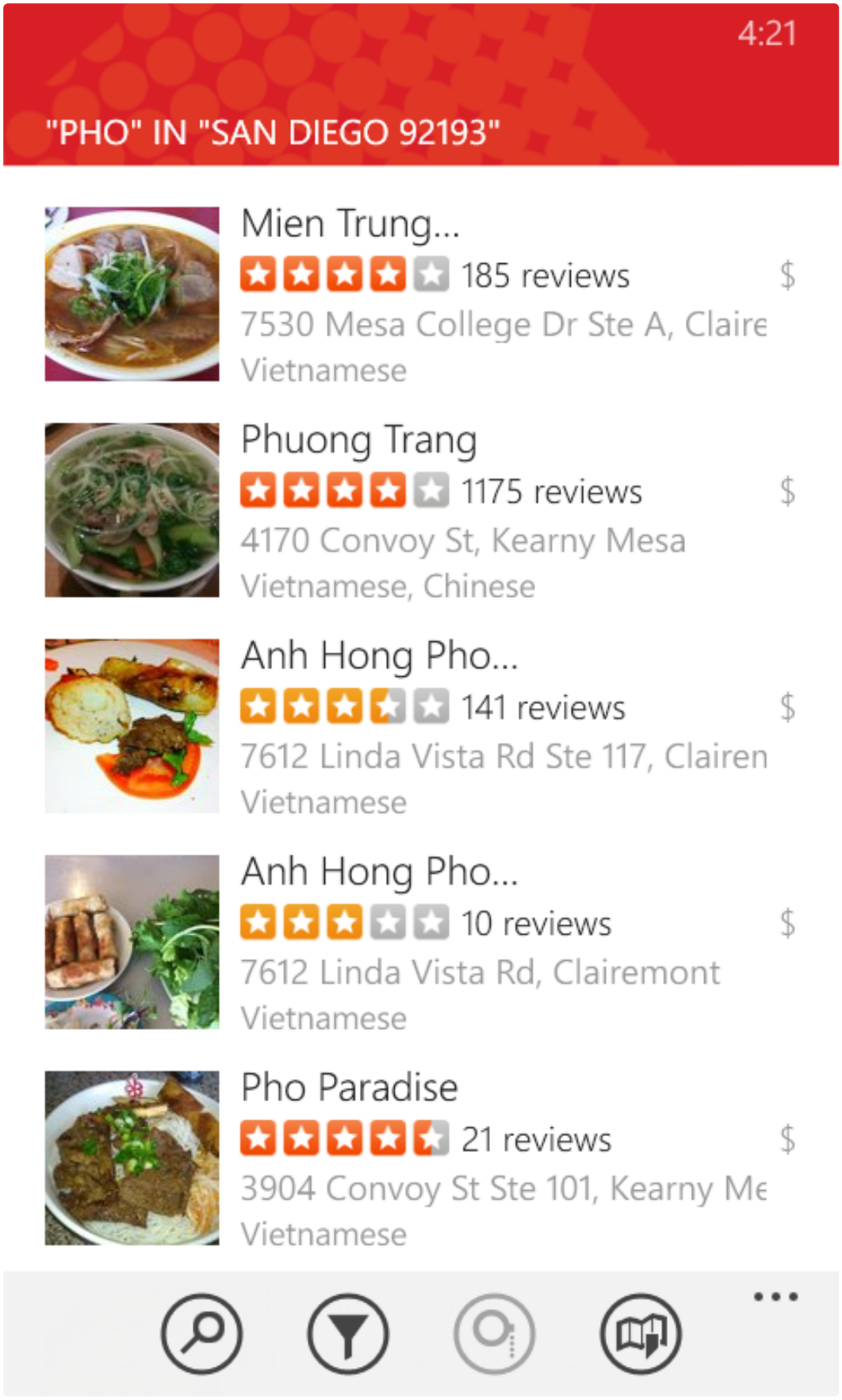
# Seeing Double On Yelp

**Tobi O., Software Engineer**
May 27, 2015

Being able to easily find what you want on Yelp is a critical part in ensuring the best user experience. One thing that can negatively affect that experience is displaying duplicate business listings in search results, and if you use Yelp often enough, you might have run into duplicate listings yourself.

We constantly receive new business information from a variety of sources including external partners, business owners, and Yelp users. It isn't always easy to tie different updates from different sources to the same business listing, so we sometimes mistakenly generate duplicates. Duplicates are especially bad when both listings have user-generated content as they lead to user confusion over which page is the "right" one to add a review or check-in to.




The problem of detecting and merging duplicates isn't trivial. Merging two businesses involves moving and destroying information from multiple tables which is difficult for us to undo without significant manual effort. A pair of businesses can have slightly different names, categories, and

addresses while still being duplicates, so trying to be safe by only merging exact matches isn't good enough. On the other hand, using simple text similarity measures generates a lot of false positives by misclassifying cases like:

- two businesses that are part of the same chain and are located close to one another
- one business that is a sub-business of another (e.g. Monterey Bay Aquarium and Jellies Experience at the Monterey Bay Aquarium)
- a professional and the practice that they work at (e.g. Coldwell Banker and Rose Parmelee - Coldwell Banker)

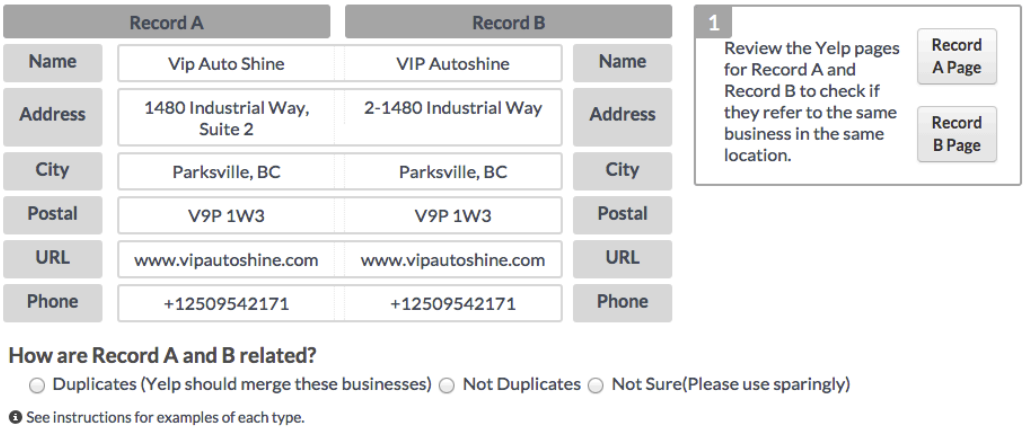
## Business Match

The first step in our deduplication system is our Business Match service. Using a wrapper over Apache Kafka, every time a new business is created or a business's attribute is changed, a batch that consumes messages published to the new_business and business_changed topics calls Business Match to find any potential duplicates of the affected business above a relatively low confidence threshold. Business Match works by taking partial business information (such as name, location, and phone) as input, querying Elasticsearch, reranking the results with a learned model, and returning any businesses that the model scores above a scoring threshold. If Business Match returns any results, the business pairs are added to a table of potential duplicates.

Our User Operations team is responsible for going through pairs of businesses in the table and either merging them or marking them as not duplicates. However, the rate at which duplicates are added to the queue far outpaces the rate that humans can manually verify them which motivated us to develop a high-precision duplicate business classifier that would allow us to automatically merge duplicate pairs of businesses.

## Getting Labelled Data

In order for our classifier to work, we needed to get thousands of instances of correctly labelled training data. For this, we sampled rows from our business duplicate table and created Crowdflower tasks to get crowdsourced labelings. We've launched public tasks as well as internal-only tasks for our User Operations team which let us easily create a gold dataset of thousands of accurately labelled business pairs. In the future, we are planning on trying an active learning approach where only data that our classifier scores with low confidence is sent to Crowdflower, which would minimize the amount of necessary human effort and allow our classifier to reach a high accuracy with a minimal number of training instances.




## Features

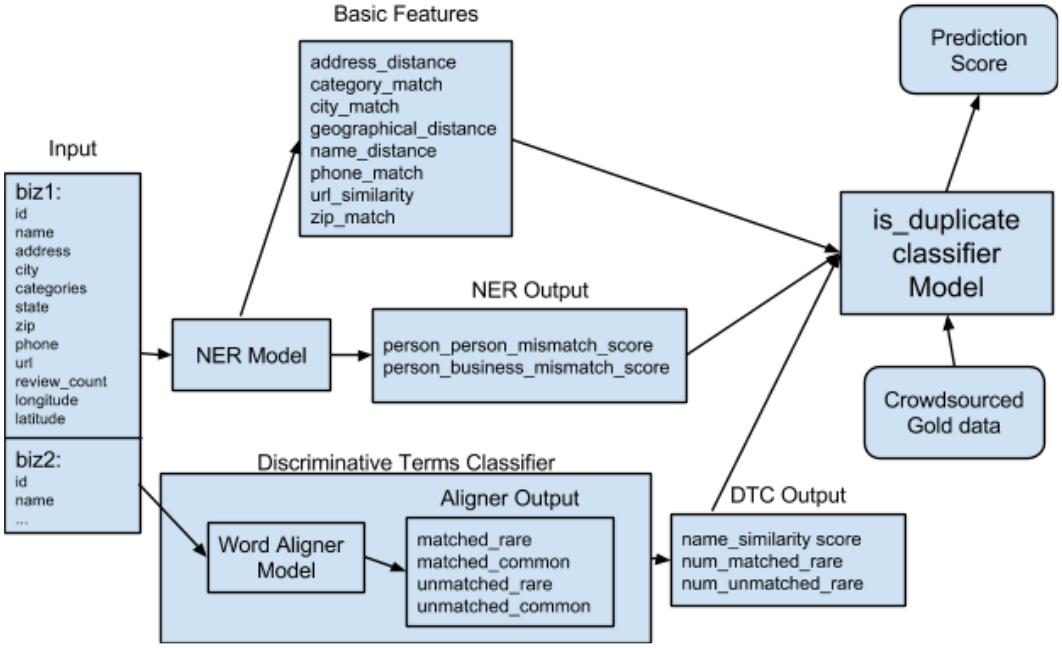
Our classifier takes as input a pair of businesses and generates features based on analyzing and comparing the business fields. It uses the same model (scikit_learn's Random Forests) and many of the same basic features as Business Match like geographical distance, synonym-aware field matching, and edit distance / Jaccard similarity on text fields. In order to capture the kinds of false positives described earlier, we also added two intermediate classifiers whose output was used as features for the final classifier.

We created a named entity recognizer to detect and label business names that indicate a person (e.g. lawyers, doctors, real estate agents) in order to detect the differences between a professional and their practice or two professionals working at the same practice.

Another feature we added is a logistic regression classifier that works by running a word aligner on both business names, finding which terms occur on one or both business names, and determining how discriminative the similarities and differences between the two names are. It outputs a confidence score, the number of uncommon words that appeared in one name but not the other, and the number of uncommon words that appeared in both names, which are used as features in the duplicate classifier.

```
# clf = sklearn.linear_model.LogisticRegression
# significant_terms = set of terms appearing more than n times in training
def classify(left_name, right_name):
"""
    Classifies names using delta term analysis.
    :return:
        A tuple (p_is_duplicate, exact_match_rare_terms, one_side_rare_terms).

        * p_is_duplicate is the score from the log-linear classifier. It's
          probably the most relevant signal.
        * exact_match_rare_terms is the number of terms that are RARE and
          exact matches. This is generally a positive signal.
        * one_side_rare_terms is a list of terms that are RARE and either
          only occur on one side, or are partial matches. This is generally
          a negative signal.
    :rtype: (float, bool, bool)
    """
# calls an word aligner on the two names. Returns a list of
# ExactMatchTerm(string)/OneSideTerm(string) objects
    delta_terms = get_name_delta_terms(left_name, right_name)

    significant_delta_terms = [term for term in delta_terms if term.word in significant_terms]

    (p_false, p_true), = clf.predict_proba(feature_vectorize(significant_delta_terms))

# Signals outside of the classifier prediction
    exact_match_rare_terms = sum(
int(term.is_rare() and isinstance(term, ExactMatchTerm))
for term in significant_delta_terms
    )
    one_side_rare_terms = sum(
int(term.is_rare() and isinstance(term, OneSideTerm))
for term in significant_delta_terms
    )

return p_true, exact_match_rare_terms, one_side_rare_terms
```

**classify.py** hosted with ❤ by **GitHub** **view raw**




## Evaluation

Since merges are hard to undo, false positives are costly so the focus of our classifier was on precision rather than recall. Our main evaluation metric was $F_{0.1}$ score, which treats precision as 10 times more important than recall. With all of our classifier's features, we achieved a $F_{0.1}$ score of 0.966 (99.1% precision, 27.7% recall) on a held-out data set, compared to a baseline $F_{0.1}$ = 0.915 (97.1% precision, 13.4% recall) for the strategy of only merging exact (name/address) matches and $F_{0.1}$ = 0.9415 (96.6% precision, 26.4% recall) using only the basic Business Match feature set.

## Future Work

With the work done on our duplicate classifier and automatic business merging, we've been able to merge over 500,000 duplicate listings. However, there's still room for improvements on deduplication. Some things slated for future work are:

- language and geographical area-specific features
- focusing deduplication efforts on high-impact duplicates (based on number of search result impressions)
- extracting our named entity and discriminative word classifiers into libraries for use in other projects With the improvements to our classifier, we hope to be able to detect merge all high confidence duplicate business listings and minimize the necessary amount of human intervention.



Back to blog