Innovaccer Intern

Analytics Assignment.

Sachin

Problem Statements

- Variation in names leads to difficulty in identifying a unique person and hence deduplication of records is an unsolved challenge.

- The problem becomes more complicated in cases where data is coming from multiple sources.

- Train a model to identify unique patients in the given sample dataset.

Tools and Libraries Used

1. Python 3.5.4 and Ubuntu Terminal

2. Pandas Module :- For Data Management and Representation.

3. Dedupe Module:- Python Library used for Machine Learning related operations.

4. OS Module:- For Operating System related operations.

Approach Used :- Depude Library

How it works:-

Using advanced machine learning and statistics, Dedupe.io learns the best way to identify similar records in any dataset

What are Matching Records:-

first name	last name	address	phone
bob	roberts	1600 pennsylvania ave.	555-0123
Robert	Roberts	1600 Pensylvannia Avenue	

However, I bet it would be pretty hard for you to explicitly write down all the reasons why you think these records are about the same Mr. Roberts.

Approach Cont.

Record Similarity:-

One way that people have approached this problem is by saying that records that are more similar are more likely to be duplicates

The default way that we do this in Dedupe.io is to use what's called a string metric. A string metric is an way of taking two strings and returning a number that is low if the strings are similar and high if they are dissimilar.

One famous string metric is called the Hamming distance. It counts the number of substitutions that must be made to turn one string into another

For example, `roberts` and `Roberts` would have Hamming distance of 1 because we have to substitute `r` for `R` in order to turn `roberts` into `Roberts`

There are lots of different string metrics, and we actually use a metric called the Affine Gap Distance, which is a variation on the Hamming distance.

Approach Cont.

Setting weights and making decisions

Let's say we calculated the **record distance** and we found that it had a value of **8**. That number, by itself, is not that helpful.

Ultimately, we are trying to decide whether a pair of records are duplicates. Does an 8 mean that the pair of records are really similar or really far apart, likely or unlikely to be duplicates?

Also, we really would rather not have to set the weights manually every time and it can be very tricky to know which fields are going to matter.

we can solve both problems with a technique called **regularized logistic regression**.

If we supply pairs of records that we label as either being duplicates or distinct, then Dedupe.io will learn a set of weights such that the record distance can easily be transformed into our best estimate of the probability that a pair of records are duplicates.

Approach Contd.

Active Learning

I order to learn the field weights, Dedupe.io needs example pairs with labels. Most of the time, we will need people to supply those labels. To reduce the amount of time that labeling takes, we use an approach called **active learning**.

With active learning, Dedupe.io keeps track of unlabeled pairs and their currently learned weights. At any time, there will be a record pair Dedupe.io will believe have a near a 50% chance of being a duplicate or distinct.

By always asking you to label the record pair Dedupe.io is least certain about, we will learn the most we possibly can about your dataset for each training pair.

Once a record pair is labeled, Dedupe.io automatically relearns the field weights.

With these new weights, there will now be a different record pair that Dedupe.io is most uncertain about, and that’s the next one you’ll be asked to label.

Approach Cont.

Making Smart Comparison

How long would it take to duplicate a thousand records?

Within a dataset of thousand records, there are (1,000 x 999) / 2 = 499,500 unique pairs of records. If we compared all of them using our magic tool it would take **six days**.

If we want to work with moderately sized data, we have to find a way of making fewer comparisons.

Duplicates are Rare

In real world data, nearly all possible pairs of records are not duplicates.
In the example below, two pairs of records are duplicates—(1, 2) and (3, 4), while there are four unique pairs of records that are not duplicates—(1,3), (1,4), (2,3), and (2,4).

Typically, as the size of the dataset grows, the fraction of pairs of records that are duplicates gets very small very quickly.
If we could only compare records that were **true duplicates** we would not run into the explosion of comparisons. Of course if we already knew where the true duplicates were, we wouldn't need to compare any records.

Approach Cont.

Blocking

Duplicate records almost always share some *thing* in common. If we define groups of data that share some thing and only compare the records in that group, or block, then we can dramatically reduce the number of comparisons we will make.

This task is called blocking, and we approach it in two ways: **predicate blocks** and **canopies**.

Predicate Blocking

A predicate block is a bundle of records that all share a feature - a feature produced by a simple function called a **predicate**.

Predicate functions take in a record field, and output a set of features for that field. These features could be “the first 3 characters of the field,” “every word in the field,” and so on. Records that share the same feature become part of a block.

Index Blocking

Dedupe.io also uses another way of producing blocks from searching and index. First, we create a special data structure, like an inverted index, that lets us quickly find records similar to target records. We populate the index with all the unique values that appear in field.

For each record we search the index for values similar to the record’s field. We block together records that share at least one common search result.

Combining Block Rules

If it’s good to define blocks of records that share the same city field, it might be even better to block records that share BOTH the city field AND zip code field. Dedupe.io tries these cross-field blocks. These combinations blocks are called **disjunctive blocks**.

Approach Cont.

Grouping Duplicates

Once we have calculated the probability that pairs of record are duplicates or not, we need to transform pairs of duplicate records into clusters of duplicate records. Three, ten, or even thousands of records could all refer to the same entity (person, organization, ice cream flavor, etc.,) but we only have pairwise measures.

Let's say we have measured the following pairwise probabilities between records A, B, and C. The probability that A and B are duplicates is 60%, the probability that B and C are duplicates is 60%, but what is the probability that A and C are duplicates? Let's say that everything is going perfectly and we can say there's a 36% probability (60% x 60%) that A and C are duplicates. We'd probably want to say that A and C should not be considered duplicates.

In the end, we found that **hierarchical clustering with centroid linkage** gave us the best results. This algorithm treats all points within some distance of centroid as part of the same group. In this example, B would be the centroid - and A, B, and C would all be put in the same group.

Unfortunately, a more principled answer does not exist because the estimated pairwise probabilities are not transitive. Clustering the groups depends on us setting a threshold for group membership—the distance of the points to the centroid. Depending on how we choose that threshold, we'll get very different groups, and we will want to choose this threshold wisely.

Approach Cont.

Choosing a good threshold

Dedupe.io can predict the probability that a pair of records are duplicates. So, how should we decide that a pair of records really are duplicates? The answer lies in the tradeoff between precision and recall.

As long as we know how much we care about precision vs. recall, we can define an F-score that will let us find a threshold for deciding when records are duplicates that is optimal for our priorities.

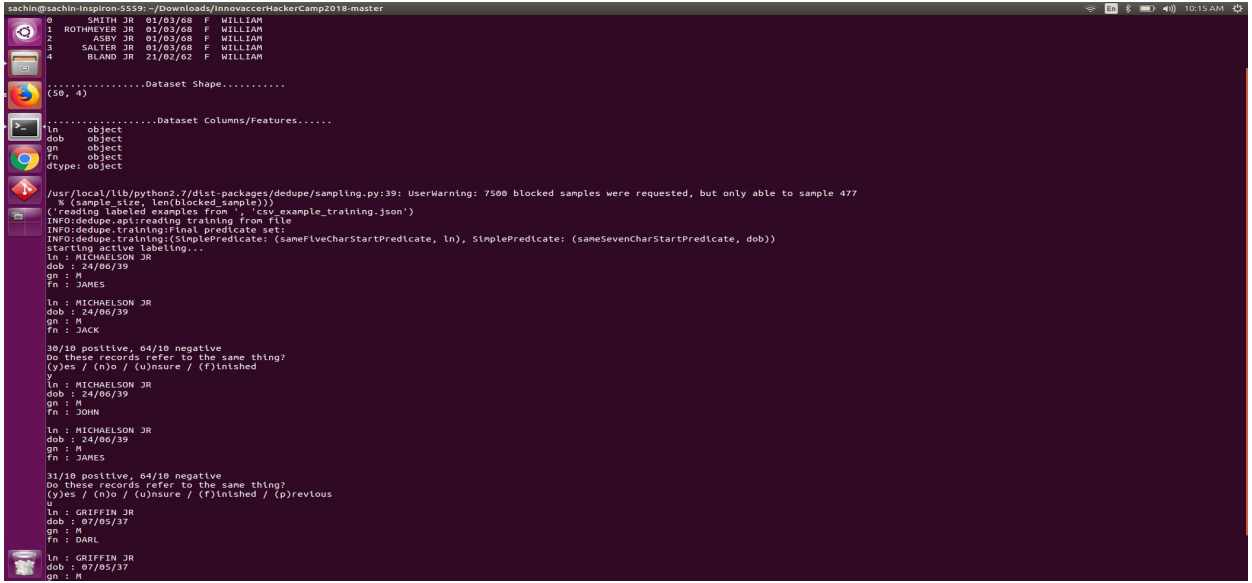
Typically, the way that we find that threshold is by looking at the precision and recall of some data where we know the real duplicates. However, we will only get a good threshold if the labeled examples are representative of the data we are trying to classify.

So here's the problem - the labeled examples that we make with Dedupe.io are not at all representative, and that's by design. In the active learning step, we are not trying to find the most representative data examples. We're trying to find the ones that will teach us the most.

The approach here is to take a random sample of blocked data, and then calculate the pairwise probability that records will be duplicates within each block. From these probabilities we can calculate the expected number of duplicates and distinct pairs, so we can calculate the expected precision and recall.

Results

Following is the screenshot of the result:



References:

https://www.druva.com/blog/3-ways-dedupe-duplicated-duplicates/

https://en.wikipedia.org/wiki/Precision_and_recall

https://dedupe.io/