Master Data Management - report

Personal Information

Name: Catalin-Stefan Tiseanu
Topcoder handle: CatalinT
Email-address: ctiseanu@gmail.com
Final submission: sub22.csv (output only task)

Approach Used

SUMMARY

The task at hand was about data reduplication, i.e identifying the rows of a database (in this case a cvs file) which are similar. My algorithm has 4 stages:
(1) Clean the initial cvs's and generate additional columns, such as last name, zip code or gender.
(2) Use blocking, a technique used to group the initial rows into buckets such that it is feasible to directly compare all rows in a given bucket.
(3) Construct similarity features for each pair of rows from (2), such as similarity of their last name, distance in taxonomy between their names and so on.
(4) Use a supervised machine learning algorithm to train on the labeled dataset from (3) in order to be able to predict on the test set.

APPROACHES CONSIDERED

From the start, I considered a supervised machine learning approach based on selecting an initial candidate set of pairs via blocking, and the applying an xgboost model on the result data.

I also studied other pen source technologies, most notably FRIL. The alternative was to used a weighted similarity score (i.e if the street address matches, add 25 points to the score) and then used a cut-off to decide whether a pair was duplicate or not (i.e. if the score >= 60 points, than they are duplicates). The obvious disadvantage of this technique was that it didn't take the training data into account. Additionally, the possibilities for blocking keys were limited.

APPROACH ULTIMATELY CHOSEN

I used the approach described in the summary.

STEPS TO APPROACH ULTIMATELY CHOSEN

I split my code in 4 main modules, which I will describe in detail below:

(1) **solution.py** - contains the actual logic glueing everything together

(2) **name_recognizer_module.py** - contains the logic for deciding whether it is a person or an organization name, and if it's a person, also computes the prefix (if present), suffix (contain credentials such as MD), first name, last name and gender (based on first name)

(3) **tag_module.py** - deals with health care provider taxonomy and computes similarity of tags based on that (i.e how closely related are ‘Internal Medicine’ and ‘Cardiology’ ?).

(4) **ml_module.py** - deals with trainining the machine learning algorithm (using xgboost) on the training set and predicting on the testing set.

(5) additionally, **nip_processing_module** shows the processing steps made on the nip data.

(6) additionally, a simple **utils.py,** which is self-explanatory

In detail:

(1) Things start off with ***process_data***, which takes the initial data cvs (either training or test) and computes additional features based of it, such as category, first name, last name, gender, prefix set, suffix set, street address, city, state, zip code and taxonomy tags. Then, a set of candidate pairs to check for being duplicates is formed via blocking. This is done in *compute_blocking,* using the cvs’s generated in the previous steps. Specifically, rows are assigned to buckets in 2 ways, by using the **min name** form as the hash key. Min name means that only the set of letters in a string matters. This is achieved by sorting the string and removing punctuation marks. For example ‘john’ becomes ‘hjno’. This enables it to be assigned in the same bucket as ‘jnoh’, since both contain the same sets of letters. This was made after observing that the huge majority of misspellings are done not by erasing or inserting a letter, but by rather swapping letters. Finally, the min name form is used for the name (after the preprocessing step) and for the street address. This generates a large number of candidate pairs which need to be further examined by a machine learning algorithm. Before that, features for each candidate pair need to be generated.
This is done in ***compute_regression_data***. It has two modes, according to whether the labels are known (for training data), or not. Given that this generates a large number of candidate pairs for training, only 20 x number duplicates are kept as negatives, i.e all positives are kept (duplicates) and 20 x that negatives. This is made in order to speed processing time.
Each pair of candidate pairs is turned in a feature vector representing their similarity. The features generated are:

- ”name_match_jw”, “street_match_jw”, “address_match_jw” - how close the strings representing the name, street address and whole address are in terms of jar winkler distance
- ”last_name_match”, ”gender_match”, “first_name_match” - do these fields match ?
- “match_name_jaccard” - the name is split into tokens (via whitespace), and a Jaccard similarity score is computed on the resulting two sets (one for each candidate row in the pair)
- ”min_name_tokens”, “max_name_tokens” - the minimum and maximum number of tokens present in the previous step
- ”prefix_match”, ”suffix_match” - how well to the prefix sets and suffix sets match (via Jacquard again)
- “mi_match” - are the middle initials of one row equal or contained in the other one ?
- ”match_state_code”, ”match_po_code”, “match_city” - boolean variables representing whether the state code, zip code and city match
- ”category”, “same_category”- features representing whether the categories of the two rows match

- “same_dot” - do both rows have a dot in their suffix (such as M.D.)
- ”match_tags”, ”mean_match_tags”, “tags_best_dist” - computes the best match for the tags, mean score for the match and the single best distance between tags.
- ”pair_1”, ”pair_2”, “label” - the id of the first row, the id of the second row, and the label (duplicate or not)

Finally, this feature data is fed into a machine learning algorithm for fitting and predicting. Note that now the problem has turned into a standard labeled machine learning problem.

(2) Basically works just as in the description above. ***__init__*** initializes the first name→gender mapping, the set of valid suffixes and valid prefixes. ***categorize_name*** takes name (column 1 in the csv’s) and decides whether it is a person name (category 0) or a company name (category 1). If it is a category 0, in ***construct_human_form*** it computes the first name, last name, prefix set (such as Dr or Prof) and suffix set (such as MD DO) and gender (0 for male, 1 for female or 2 for unsure).

(3) The module starts by taking a raw dump from http://www.wpc-edi.com/reference/ and computes a more readable form, in ***process_raw_taxonomy***. Then, in ***process_tree_taxonomy***, it takes the enhanced format and turns it into a taxonomy tree. The relevant files and be found in the folder ***taxonomy***. Finally, **process_raw_taxonomy** binds everything together. These steps are done in a processing step, before even looking at the training data.

On the feature computations side, **match_tags** takes 2 tags (present in the last column in the cvs’s) and computes how closely they match, with a bigger number being better. Specifically, 3 - exact match, 2 - one tag begins or ends with the other, 1 - one tag is a child of the other, 0 - no similarity.

(4) This module performs a standard machine learning pipeline with 2 twists. Specifically

1. the feature data is loaded as the *dataset*
2. add additional candidate pairs to both the training set and to the testing. The blocking key used is very restrictive (first 2 tokens from the name + the state code + tag set)
3. the *dataset* is split into 80% training, 20% testing (testing obviously used only to gauge the training efficacy)
4. an xgboost model is trained for the training data for 30 iterations, with max tree depth 4 and eta 1.0. It used the binary:logistic classifier loss function
5. the model is then used to predict the testing feature data (made from the initial test.cv)
6. finally, transitive closure is used to detect the set of pairs which were not considered initially, but should be according to the prediction. For example if (a, b) is predicted to be a duplicate, and (b, c) is predicted to be a duplicate, then edge (a, c) is added to the set of candidate pairs. The resulting set is the scored again using the trained xgboost model.
7. finally, the initial results are concatenated with the results based on the newly added set after the transitive step and both are written to the final ***submission.csv*** file

OPEN SOURCE RESOURCES AND TOOLS USED

I made heavy use of Python and related python machine learning libraries. Specifically:

- Python
- Numpy, pandas, sklearn for data processing
- Networkx for graph processing
- Xgboost for a highly performant gradient boosting library (https://github.com/dmlc/xgboost)
- IPython for rapid iteration in the notebook (not included in the final deliverable)
- Jellyfish for string similarity metrics (such as Jaro-Winkler distance)
- Termcolor for colorful printing to the console
- Multiprocessing for well, multiprocessing

Additionally, I used the following open source text resources:
NPI files - http://nppes.viva-it.com/NPI_Files.html
Health Care Provider Taxonomy Code Set (this is linked directly by the nppes site) - http://www.wpc-edi.com/reference/

ADVANTAGES AND DISADVANTAGES OF THE APPROACH CHOSEN

The main advantage of the chosen approach (selecting an initial set of pairs to directly compare) is that it enables an supervised approach which has high accuracy (95%+). Additionally, by tuning the blocking algorithm used to select less pairs to be directly compared the running time can be decreased (at the expense of accuracy).

The biggest disadvantage of the algorithm is that pairs which are not put in into the same bucket by the blocking algorithm will never be examined (they don't stand a chance to be considered duplicates so to say). On the training set, ~4000 duplicates out of ~125.000 were not considered in the prediction phase (aka scoring by the machine learning algorithm). The pairs not considered obviously contributed the full error (1.0) to the Brier score.

COMMENTS ON LIBRARIES

I used a fairly standard Python stack for machine learning. The more unusual addition is Xgboost, which is not that well known as a machine learning library, even though it helped win several other international machine learning competitions.
Xgboost has the advantage (over sklearn for example) that is highly optimized, speeding the training time considerably.

COMMENTS ON OPEN SOURCE RESOURCES USED

The main open source resource used is the nip provider database. This has 4.8 million rows similar to the ones given in the training and testing cvs's, only with better info (i.e more columns, such as first name, last name, gender and so on). I used this to extract 2 things:

(1) A first name to gender mapping based on the occurrences of each (first name, gender) pair in the nip dataset. First names for which there wasn't an overwhelming proportion of one gender to the other were marked as unsure (such as 'Kelly' for example).

(2) Extract the set of the possible medical suffixes, such as MD, CRNA, DO, etc. from the 'credentials' column of the dataset.

Obviously, such a rich dataset would have had other uses, but unfortunately I didn't have time to explore them.

Related to the nip dataset, I used the healthcare provider code set taxonomy

SPECIAL GUIDANCE GIVEN TO ALGORITHM BASED ON TRAINING

The maximum tree depth was fine-tuned. Also, another restricitve blocking key which would give a large ratio of positive to negative rows was developed.

POTENTIAL IMPROVEMENTS TO YOUR ALGORITHM

The main improvement to the algorithm is to consider more blocking keys in order to include as many as possible of the 'missing duplicates' described in the advantages and disadvantages section. Special care must be given in order to not include too many negatives duplicates.

For example, in extremis, by considering all pairs (obviously this wouldn't run in time, but just as an example), we are going to include all duplicates (~125.000 for the training set), but also a lot of negatives (~230.000.000.000) which would significantly dilute the capacity of the machine learning algorithm to separate positives from negatives.

Several other blocking keys were considered (such as first name, city) but they suffered from the problem mentioned above (only including 1000 duplicates amongst 1.000.000 negatives, for example).