Louis Massera

Entity Resolution Technique

Goal : Find pairs of id that represent the same restaurant in two different data files.
Main idea to solve the problem : build a classification model that, given a similarity vector between one element of the first file and one element of the other one, can tell if those two elements are similar.

To do so, we must build, from *locu_train.json*, *foursquare_train.json* and *matches_train.csv*, a training matrix of similarities. Theoretically, since both *locu_train.json* and *foursquare_train.json* contain 600 elements, we would have to consider 360000 possible matches. However we will use some heuristics to reduce the size of our matrix afterwards.

- Preprocessing :

  - Data cleaning :

From original files, we keep all features except «country», «locality» and «region» since we assume all restaurants are in New York. Of course it would be really important to consider them if it was not the case. To perform valuable comparisons between fields, we clean them and make sure that they have the same format («phone» for instance).
There are three functions in *Entity_Resolution.py* that take care of most strings transformations :

- str_cleaning : get rid of special characters.
- phone_cleaning : put foursquare « phone » in the same format as locu «phone».
- website_cleaning : extract the website name.

«address» field is parsed using *usaddress* package. We only keep four resulting fields, based on the percentage of existing values they contain : «AddressNumber», «StreetNamePreDirectional», «StreetName», «StreetNamePostType». All fields are cleaned using str_cleaning.

Next step is building the matching matrix, containing all possible combinations.
We can move on to creating features that measure the similarity between our elements.

- Feature Engineering :

Using the longitude and latitude, we can compute the distance (dist) in kilometers (using *vincenty* package).
Then, we create a feature that will check if two fields are similar (same name, same phone number, same website, for instance). The function check_equality in *Entity_Resolution.py* performs it.
We can also consider the levenshtein distance (leven) and the jaro-winkler distance (jw) between two fields, with *jellyfish* package. The compute_distances function in *Entity_Resolution.py* creates those features.

In the end, we have built 16 features :
'dist', 'leven_phone', 'jw_phone', 'leven_name', 'jw_name', 'leven_street_name', 'jw_street_name', 'leven_address', 'jw_address', 'same_postal_code', 'same_street_number', 'same_address', 'same_website', 'same_phone', 'same_name', 'same_street_name'.

- Avoid Pairwise Comparison :

To avoid pairwise comparison, we filter on the features we have just created. We only keep the similarity vectors that have a distance that is less than 1 or at least one of their «same» fields that evaluates to True (1). From 360000 possible matches in the training set we move to less than 35000. From 160000 in the test set we get less than 16500.

- Imbalanced Data :

However, our data is strongly imbalanced, about 1% of it is labeled as a match.
To deal with that problem, we first fill the missing values of each feature with the mean. Using KNN method would give better results but *fancyimpute* package did not work well.
Then, by using SMOTE (Synthetic Minority Over-sampling Technique), we can obtain a more balanced dataset. This is helpful to get a better recall, without affecting the precision if used wisely.

- Model Selection :

We first performed a GridSearch on a gradient boosting classifier using X_train only (similarity matrix containing the features engineered on the train files.) to determine the best hyper-parameters. *xgboost* package is used here.
Then we used the whole training samples and the best hyper-parameters to build the final model.

- Feature Importance :

Our model gives the importance of each feature. Top five features are : 'jw_name', 'dist', 'leven_address', 'leven_name', 'same_phone'.
It seems that all type of features contributed to get a good prediction. We could try to make our model sparser, or design new features, but our result is satisfactory.

- Results (precision, recall, F1 score) :

Precision = 100% / Recall = 96.67% / F1 score = 98.31%