Match prediction enhancement for dating applications

Project update

Authors - Marco Cavenati, Mayank Narang, Ilaria Pilo

Motivation

With the pandemic, the usage of dating applications has seen a rapid increase. However, users often struggle to find like-minded people with whom to build a durable and successful relationship. To overcome these limitations, we aim to develop a machine learning model that allows the system to propose higher quality matches between users in less time.

Dataset overview

We are using the Speed Dating | Kaggle dataset, which stores the results of a series of speed dating events. The dataset contains the information collected before, during and after the event, for a total of **195 columns**. The total number of recorded interactions is **8378**. Alongside with the dataset, we have at our disposal the original survey given to the people who participated in the event, where we can retrieve the semantics of the columns present in the dataset.

Cleaning the dataset

The two biggest and time consuming issues faced with the dataset were:

1. Analyse and understand the semantics of the couluns to understand if the same feature could be retreived in the dating app context in the registration phase.
2. Manage the inconsistency between the survey methodologies used in the different waves of the event.

After the selection process, only 57 features are left. Here, we describe some of them:

- **match**: match (label of the model)
- **iid**, **pid**: id of the person participating in the event; partner id
- **gender**, **age**, **race**: gender, age and ethnicity (coded) of the person
- **imprace**: How important is it for the person that their partner is of the same racial/ethnic background (on a scale of 1-10)
- **field_cd**, **career_c**: Field of study, Intended career (coded)
- **goal**: primary goal in participating in the event (coded)
- **go_out**, **date**: How frequently the person goes out generally, and specifically on dates (coded)

Our dataset also includes:

- information on activities the person finds interesting, on a scale of 1-10 (such as playing sports, dining out, hiking, dancing, reading, music...).
- person's opinion on attributes they believe are important in a potential date, as well as how they rate themselves according to the same attributes (attractive, sincere, intelligent, fun, ambitious, has shared interests/hobbies).

To tackle the second problem, we remove from the dataset the data collected with different criterias, since it hasn't been possible to map that data into a form consistent with the rest of the data. Also, the records containing `NA` values has been dropped.

After completing the cleaning procedure, we obtain 3982 "No match" samples and 736 "match" samples.

Reshaping the dataset

Then, we reshape the data into a suitable form for the ML algorithms. In particular, we use the `iid` and `pid` fields to join the dataframe with itself, so that each record contains both answers of the couple members. We also minimize the memory requirements by choosing optimal datatypes.

Applying one hot encoding

Since the dataset contains a vast majority of categorical features, we have to choose the proper encoding for each of them. In particular, we observe two different cases:

1. The categorical feature represents a scaling
2. The categorical feature is simply a set of values

Clearly, the two cases need to be addressed separately. While a standard encoding is appropriate to represent case (1), we want to apply one hot encoding to case (2), so that our models do not assume any scale of importance among different values.

After this final step, we obtain a total of **186** features (excluding the label `match`).

Methodology

When analysing the dataset, we first split it into training and test set (80-20 ratio). We leave the test set untouched, to avoid taking biased decisions on our models, and we feed the dataset to a grid search algorithm, implementing a 5-fold cross validation approach. This way, we are able to optimize our models' hyperparameters, and to value our data as much as possible.

Moreover, we analyse the effect of some preprocessing techniques, such as feature scaling, , feature interactions, Principal Component Analysis (PCA) and Synthetic Minority Over-sampling Technique (SMOTE).

Feature interactions

Recall that, in our dataset, each column is present in two separate instances, one for each member of the couple. We believe that our models could benefit from adding the interaction between each pair of instances, as two people will more probably match if they have many interests in common.

However, adding feature interactions further increases the dimensionality of our problem, raising the number of features to **279**. For this reason, we decided to include the possibility to drop the original features, and keep only their interactions. As a consequence, the dimensionality of our model drops to **93**.

Fighting the curse of dimensionality

As we are dealing with a high dimensional dataset, we have to take precautions against the curse of dimensionality. To do so, we choose models which are not based on the notions of distance and similarity,

such as Random Forest, Support Vector Machine and Logistic Regression. Furthermore, we try to apply k-Nearest Neighbors, to check if it performs bad as expected.

Moreover, we consider PCA as a preprocessing step. First, we use PCA to compute how many features explains the 90-95% of our dataset. We repeat the same procedure for the dataset with feature interaction (with and without drop). Finally, we use the obtained results as number of component for the PCA algorithm.

The imbalance problem

As we already noticed, our dataset is heavily imbalanced. As a consequence, we pick as metrics balanced accuracy, F1-Score and recall (as accuracy can be misleading).

Furthermore, we make use of a Stratisfied Split for the generation of the training, validation and testing sets: we ensure that each fold has the same proportion of observations no match/match.

However, these measures are not enough to obtain acceptable results from our models. For this reason, we consider to apply SMOTE, an over-sampling technique which adds synthetic points to the minority class.

Models

In order to face the classification task, our choices in terms of models have so far been the following:

- Random Forest
- k-Nearest Neighbor (for comparison purposes only)
- *Logistic Regression [work in progress]*
- *Support Vector Machine [work in progress]*

Surprisingly enough, KNN performs quite good (recall = 0.7, balanced accuracy = 0.82), as long as we apply feature scaling and we choose k = 1. Both PCA and SMOTE does not seem to have a significant impact on our results. Random Forest with SMOTE preprocessing returns similar results, with a recall of 0.65 and a balanced accuracy of 0.82.

Finally, since our datset is both highly imbalanced and dimensional, we observed very bad performances in optimization-based algorithms (LR and SVM), as they fail to converge. We plan to face these issues by further investigating the relations among some features as well as in the ways indicated in the next section.

Next steps

- Complete LR and SVM implementation; try new models, and different over-sampling techniques
- Enhance the feature interactions function, by considering different combinations of features, and proposing new feature engineering ideas knowing the features meaning in our scenario
- Modify the pipeline to support cost-sensitive training (to further reduce the impact of class imbalance)
- Implement bias-value decomposition, to better understand our models' performance

Contribution

- Marco Cavenati: dataset analysis, cleaning and features selection.
- Mayank Narang: implementation of the algorithms.
- Ilaria Pilo: data split and preprocessing, k-fold and grid search wrappers, training pipeline definition.

References

1. 10 Techniques to deal with Imbalanced Classes in Machine Learning
2. imbalanced-learn documentation
3. scikit-learn documentation