Match prediction enhancement for dating applications

Project for the MALIS course taught at EURECOM by Maria A. Zuluaga

---

Authors: Marco Cavenati, Mayank Narang, Ilaria Pilo

Introduction

With the pandemic, the usage of dating applications has seen a rapid increase [1]. 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 (ML) model that allows the system to propose higher quality matches between users in less time.

Such models will be trained on the Speed Dating | Kaggle dataset, which stores the results of a series of speed dating events. In such an experiment, participants were posed different questions about themselves and their ideal partner, with the goal of studying dating behavior [2].

However, we believe that such a dataset could be also used to predict whether a given match would be successful or not. The same questions are easily replicable in any dating application during the user's registration process. Then, answers of different people can be combined and fed to the model, which will return the predicted outcome of such couples (match / no match). With this information, the system will propose to the user other users having a `match` prediction, therefore improving the user experience.

All the code developed is available on GitHub.

Dataset overview

The dataset contains information collected before, during and after the event, for a total of 195 columns. The total number of records is 8378**,** corresponding to 4189 dates. The dataset is imbalanced — 3.5k "No match" records against ~700 "match" records — and it contains mainly categorical attributes.

Cleaning the dataset

Feature selection

We analyze the semantics of the features to decide if the same information can be retrieved in the dating app context during the registration of a new user. If not, we drop the column, since we would not be able to retrieve the information in our use case.

After the selection process, only 57 features are left:

- **match**: label of the model
- **iid**, **pid**: ids of the two involved people
- **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)
- Information on activities the person finds interesting, on a scale of 1-10 (such as playing sports, dining out, hiking, dancing, reading, music...)
- The 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).

The detailed reasons for the removal of each column are explained in the *Dataset_analysis* notebook, alongside the details of the selected features.

Record selection

The dataset presents some inconsistencies between the methodologies used in the different waves of the event. To tackle this problem, we remove from the dataset data collected with different criteria, since it has not been

possible to map it into a form consistent with the other samples. Also, all records containing `NA` values are dropped.

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 answers of both couple members. Then, we remove duplicate entries (e.g., the record with `iid=1234` and `pid=5678` carries the same information as the one with `iid=5678` and `pid=1234`, as they identify the same couple).

We also minimize the memory requirements by choosing optimal data types.

After the cleaning and reshaping phases, we obtain 1991 "No match" samples and 368 "match" samples.

Applying one-hot-encoding

Since the dataset contains 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 an **ordered set** (e.g. how often do you go out? 1 = Several times a week, 2 = Twice a week, ...)
2. The categorical feature is just a **set of values** (e.g. field of study: 1 = Law, 2 = Math, ...)

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 analyzing the dataset, we first split it into training and test set (80-20 ratio). We leave the test set untouched, to avoid making biased decisions on our models, and we feed the training set 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 analyze the effect of some preprocessing techniques, such as feature scaling, feature interactions, Principal Component Analysis (PCA) and Synthetic Minority Over-sampling Technique (SMOTE).

Corresponding feature interactions

In our dataset, each column is present in two separate instances, one for each member of the couple (e.g., features **sports_x** and **sports_y** store how much each member of the couple enjoys playing sports - this duality holds for every feature). We call such column pairs *corresponding features*. We believe that our models could benefit from adding the interaction (i.e., the product) between each pair of corresponding features, as it could be that two people will more probably match if they have many interests in common. Generally speaking, we want to compute `feat_interaction = feat_x * feat_y`.

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

Notice that, after dropping the original features, samples in the dataset become *symmetrical* (that is, couples `(a,b)` and `(b,a)` have perfectly identical samples). This property is actually desirable, since we would like a couple to be associated with the same prediction, regardless of the order with which we load their replies.

Furthermore, we can try to extract additional information from our data, under the following assumptions. Let us consider a generic interest (for example, sports):

- If two people both like sports, then they will be probably compatible (as far as sports are concerned);
- If one person likes sports, but the other is not interested in them, then they will be probably incompatible;
- If both people do not care about sports, then we cannot assume anything on their compatibility.

This behavior is different from the one of the product interaction, as we would like to reward common interests, penalize opposite interests and ignore common disinterests (while the product assigns high values to common

interests, medium values to opposite interests and low values to common disinterests). For this reason, we design a custom interaction function, which operates as shown in Algorithm 1.

Algorithm 1 Custom Interaction Function

for each $feat$ **do**
  if is_binary($feat$) **then**
    $feat_{int} \leftarrow 0$ **if** $feat_x = feat_y = 1$
    $feat_{int} \leftarrow 1$ **if** $feat_x = feat_y = 0$
    $feat_{int} \leftarrow 2$ **if** $feat_x \neq feat_y$
  else
    if is_integer($feat$) **then**
      $feat_{int} \leftarrow |feat_x - feat_y|$ **if** $feat_x > 6$ and $feat_y > 6$
      $feat_{int} \leftarrow 4 + |feat_x - feat_y|$ **otherwise**
    end if
  end if
end for

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 (RF), Support Vector Machine (SVM) and Logistic Regression (LR). Furthermore, we try to apply k-Nearest Neighbors (kNN), as a baseline reference (we expect other models to achieve a better performance).

Moreover, we consider PCA as a preprocessing step. PCA is an unsupervised technique that reduces the dimensionality of a dataset while preserving the maximum amount of information [3].

First, we use PCA to compute how many features explain the 90-95% of our dataset. We repeat the same procedure for the dataset with corresponding feature interactions, with and without drop (Figure 1 and Table 1). Finally, we use the obtained results as the number of components for the PCA algorithm.

PCA can be also used to visualize our data, by plotting the first two principal components of each sample (Figure 2). As we can see from Figure 2, we can already deduce that our classification task will not be easy, as the two classes in the dataset are completely overlapped.

Preprocessing	**90 %**	**95 %**	**100 %**
None	87	106	186
Interactions, no drop	117	143	279
Interactions and drop	57	64	93

Table 1 - Number of features explaining 90-95-100 % of our dataset, given different preprocessing strategies.

The imbalance problem

As we already mentioned, we have an imbalanced dataset, therefore we pick as metrics *balanced accuracy*, *F1-Score* and *recall* (as the use of *accuracy* can be misleading). Moreover, since we want to minimize the percentage of false negatives (that is, the percentage of `match` samples which are classified as `no_match`), we select recall as the best metric for our grid search.

Furthermore, we make use of stratified splits 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, in many cases, these measures are not enough to obtain acceptable results from our models. For this reason, we consider applying SMOTE, an over-sampling technique which adds synthetic points to the minority class. More in detail, a sample belonging to the minority class is randomly selected. Then, points are added between such a sample and its k-nearest neighbors [4-6].

We also experiment the use of different weights for the `match` and `no_match` classes during the training phase of the models: the goal is to prevent the imbalance problem by weighting the classes inversely proportionally to how frequent they are.

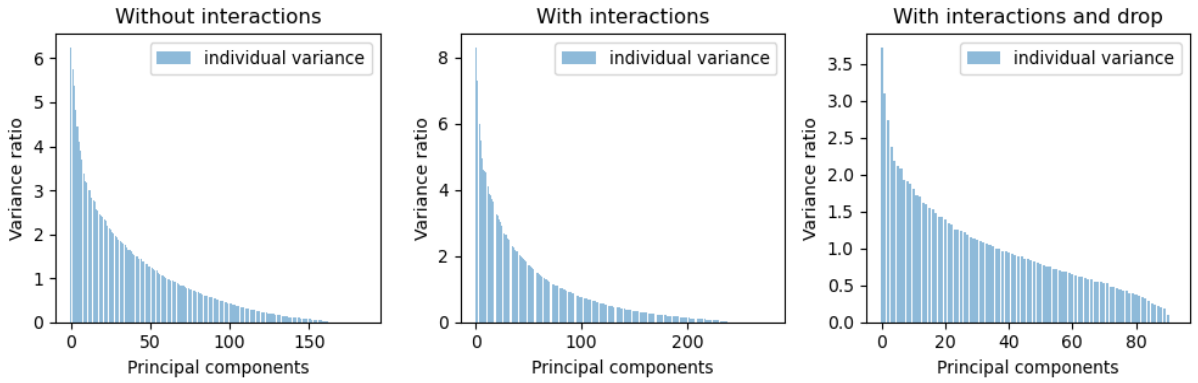


Figure 1 - Plot of variance ratio for each principal component.

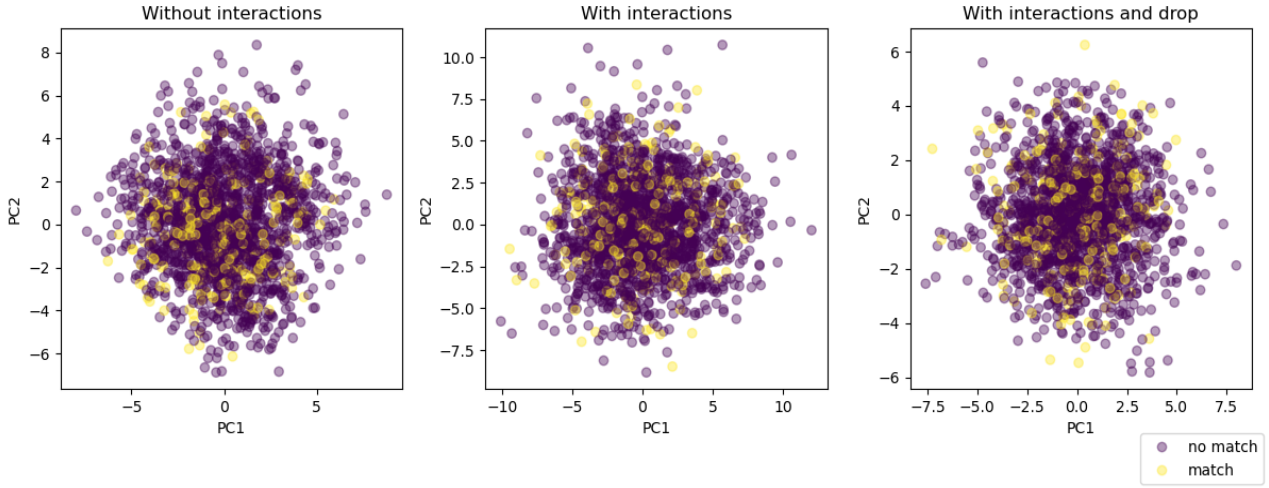


Figure 2 - Scatter plot of the first two principal components.

Models and experiments

In order to face the classification task, we try several models:

- *Standard* ML classifiers (such as RF, SVM, LR and kNN as reference);
- A custom Gaussian Mixture Model (GMM) based classifier;
- An unsupervised outlier detection technique based on Isolation Forest (IF).

Every supervised model is added to many pipelines, each including different preprocessing steps (as described in the previous section), and finally fed to the grid search function. The complete list of hyperparameters we tested, as well as the values of recall, balanced accuracy, f1 and pipeline details for the best models, are reported in Table 2.

Custom GMM-based classifier

Gaussian Mixture Models are very powerful unsupervised techniques, which assume samples were generated by a mixture of (multivariate) gaussian distributions, and compute the parameters of such distributions [7]. A GMM-based classifier fits a GMM for each class. Then, whenever a new point must be classified, the model evaluates its likelihood of belonging to each class, and uses such results to predict the label.

Since scikit-learn does not provide a GMM-based classifier (but only an estimator for the GMM parameters), we write our own scikit-learn compatible estimator [8], so that it can be included in the grid search.

Outlier detection with Isolation Forest

For this experiment, we split the dataset in training, validation and test set (0.7-0.2-0.1), and we fit the Isolation Forest, an unsupervised model for outliers detection [9], on the training set, under the assumption that all `no_match` samples have actually been generated by some probability distribution, while `match` samples are simply outliers with respect to such a distribution. Being an unsupervised approach, we cannot directly use labels during the training phase. However, we can compute the ratio of `match` samples, and use it as the value for the `contamination` hyperparameter (which stores the ratio of outliers in the dataset).

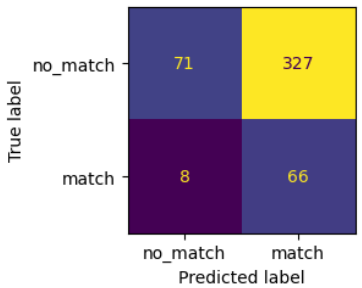
Algorithm	**Hyperparameters**	**Values**	***Best model***			
			Recall	**Balanced accuracy**	**F1**	**Pipeline**
Gradient Boosting Tree	n_estimators learning_rate max_depth min_samples_split	500 0.1, 0.05 5, 8 6	0.310	0.568	0.274	Polynomial features, SMOTE and PCA
Gaussian Mixture Model based classifier	n_components covariance_type	**1**, 2, 4, 8, 16 full, tied, **diag**	**0.897**	**0.515**	**0.271**	***No preprocessing***
k-Nearest Neighbors	n_neigbors weights	from 1 to 10 uniform, distance	0.663	0.525	0.204	Custom interactions only, SMOTE
Logistic Regression	C	10, 100, 10000	0.490	0.551	0.274	Interactions only, feature scaling
Random Forest	max_features bootstrap min_samples_split n_estimators criterion	sqrt True 2, 5 100, 200 gini, entropy	0.146	0.559	0.223	Interactions only, SMOTE, features scaling
Support Vector Machine	kernel	linear, poly, rbf	0.452	0.510	0.238	Custom interactions, SMOTE
Isolation Forest	contamination	match ratio	0.189	0.519	0.189	Interactions only

Table 2 - List of tested hyperparameters and results of the best estimators for each model.
In bold, the best overall performer, in terms of recall.

Results analysis and testing

By comparing our results, we can make the following observations:

- There is no model that returns satisfactory results - all of them are characterized by low balanced accuracy and f1 score. Hence, they are not able to maximize (positive) recall without minimizing both precision and negative recall. This is consistent with Figure 2 results: the two classes are possibly overlapped;
- kNN does not perform worse than other models. The fact that we do not gain the expected improvements when using feature-scalable algorithms further proves the low quality of our dataset;
- The IF approach performs quite poorly, meaning that the assumption it is based on is probably wrong;
- No preprocessing strategy appears to be always better than the others: different classifiers benefit from different preprocessing techniques.



Overall, the best-performing model is the GMM-based classifier with diagonal covariance, 1 component and no preprocessing. We therefore select it to classify the up-to-now-unknown test set. The obtained results in terms of recall are quite good (0.892), but they are motivated by the high bias towards the `match` label: indeed, the overall accuracy is 0.290. From the confusion matrix shown to the left, it is possible to notice the high number of false positives (that is, `no_match` samples which are wrongly classified as `match`).

Conclusions

Overall, we could not find a model able to properly solve our classification problem. Even if some classifiers achieve a good recall, their performance is not so far from the one of a dummy model, which always assigns the label `match`. The usage of variegate preprocessing strategies, of techniques to handle data imbalance and high dimensionality, of different algorithms with hyperparameters tuning and appropriate metrics did not improve the results enough. Therefore, for future work, the use of a different, tailored dataset should be considered.

Contributions

Marco Cavenati: dataset analysis, cleaning, reshape and features selection, poly and custom feature interactions.

Mayank Narang: implementation of the initial algorithms, cost-efficient training, model selection (partial).

Ilaria Pilo: data split and preprocessing, k-fold and grid search wrappers, training pipeline definition, feature interactions, Isolation Forest experiment, GMM custom classifier.

References

[1] Tinder - Match Group LLC., *The future of dating is fluid, Tinder_Future_of_Dating_3.24_FINAL.pdf (mediaroom.com)*.

[2] Raymond Fisman, Sheena S. Iyengar, Emir Kamenica, Itamar Simonson, *Gender Differences in Mate Selection: Evidence From a Speed Dating Experiment*, The Quarterly Journal of Economics, Volume 121, Issue 2, May 2006, Pages 673–697, https://doi.org/10.1162/qjec.2006.121.2.673.

[3] Wikipedia, *Principal Component Analysis*, https://en.wikipedia.org/wiki/Principal_component_analysis.

[4] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, *SMOTE: synthetic minority over-sampling technique*, Journal of artificial intelligence research, 321-357, 2002.

[5] *10 Techniques to deal with Imbalanced Classes in Machine Learning*, https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning.

[6] *imbalanced-learn Documentation*, https://imbalanced-learn.org/stable/index.html.

[7] *Gaussian Mixture Model*, https://www.techopedia.com/definition/30331/gaussian-mixture-model-gmm.

[8] *Developing scikit-learn estimators*, scikit-learn Documentation, https://scikit-learn.org/stable/developers/develop.html.

[9] *Anomaly detection using Isolation Forest – A Complete Guide*, https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/.