Scientific Computing with Python Final Report

Yi Cao, Shiqian Xu, Zhiyuan Wei

December 9, 2023

1 Introduction

In an era where the relentless pace of professional commitments, economic exigencies, and unprecedented public health challenges – epitomized by the COVID-19 pandemic – have substantially altered the fabric of social interactions, our research endeavors to navigate the intricate tapestry of human preferences and compatibility through a lens of sophisticated analytical rigor. We harness the predictive power of a Random Forest classifier to delve into a rich speed dating dataset, a repository brimming with nuanced personal attributes and preferences. This dataset serves as an exemplary arena for dissecting the multifaceted nature of matchmaking, providing us with a fertile ground for our exploratory analysis.

Our initial analytical efforts are directed towards computing compatibility scores between a specific individual and other participants, employing an array of decision trees strategically designed to evaluate various factors that could influence the probability of successful matches. To augment the precision of our matching process, we have integrated the principles of the knapsack problem, a cornerstone of optimization theory renowned for its application in resource allocation challenges. This methodological synergy allows us to impose realistic constraints such as age and the number of potential matches, reflecting typical limitations encountered in speed dating scenarios.

The essence of our approach is not merely to identify possible matches but to curate a select cohort that embodies true compatibility. Through meticulous computational analysis and strategic optimization, we have refined the pool of candidates down to ten meticulously chosen individuals. This collective represents a harmonious fusion of algorithmic accuracy and the subtleties of human interaction, thus aligning with the unique standards that individual A seeks in a partner.

In our study, we initially utilized a Random Forest algorithm to determine match rates, which subsequently served as value parameters in our application of the knapsack optimization problem. Accompanied by predetermined constraints on the number of individuals and age range, we successfully curated an optimal group of ten participants for the speed dating cohort. We leveraged the knapsack algorithm to optimize for match rates, ultimately yielding a select group that met our established criteria.

This methodical approach allowed us to not only predict but also enhance the matchmaking process, culminating in a well-defined assemblage poised for speed dating. The

employment of the knapsack problem in this context facilitated a focused optimization of potential matches, ensuring that the final selection aligns with the desired characteristics and constraints.

2 Proposed Methods

2.1 Data Preprocessing

We used the pandas to organize and clean the data, eliminating unnecessary information and filling in gaps. We then extracted 9 features, *gender*, *samerace*, *field*, *like*, *met*, *age_diff*, *sincere_o*, *ambitious_o*, *funny_o*, and separated out the label *match* from the dataset.

2.2 Random Forest

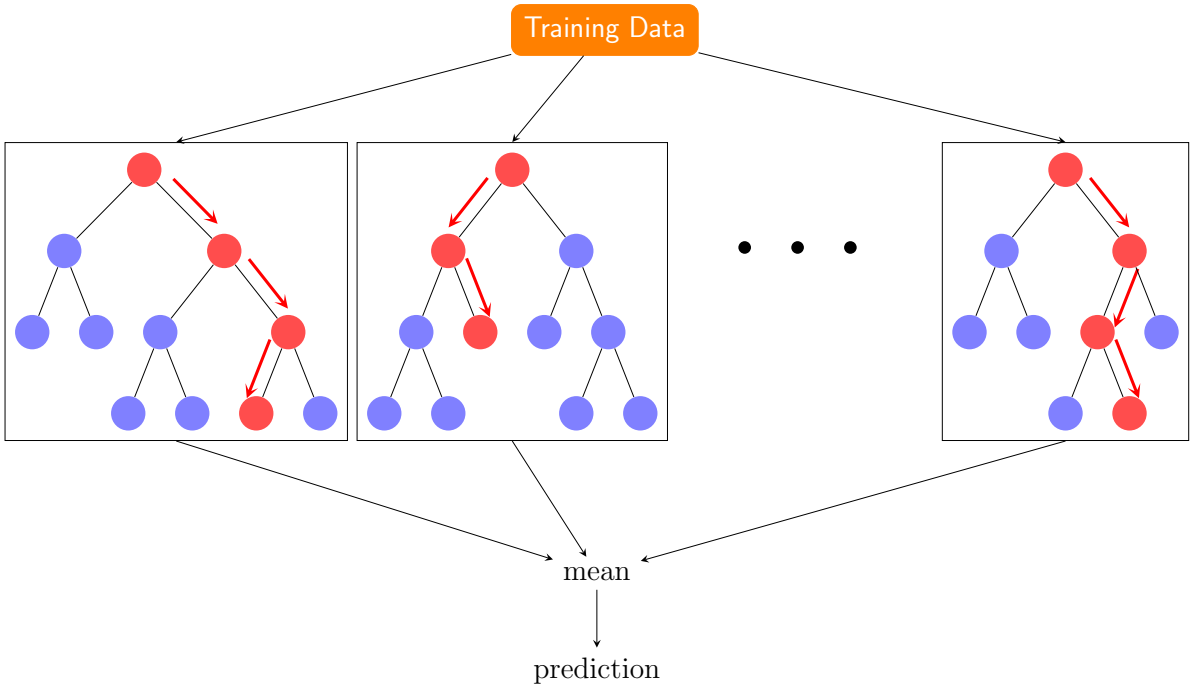
The Random Forest algorithm is a robust ensemble predictor that constructs a multitude of decision trees. Each tree within the ensemble is grown on a distinct bootstrap sample drawn from the original dataset D. Bootstrapping, a method of random sampling with replacement, introduces variability by providing different subsets of data for each tree, hence embedding the "random" aspect into the model.

The randomness is further emphasized during the tree growth process at each node within a tree, where a random subset of F features is selected to determine the best split. This approach, which increases diversity among trees and mitigates overfitting, often sets F to be the square root of the total number of features for regression tasks.[1]

For making predictions, a new data instance is passed through each tree. The final prediction for regression is determined by averaging the outputs of all trees, which captures the central tendency of the ensemble. In classification tasks, a majority vote system is employed to select the most frequent outcome from all trees as the final prediction.[2]

In our study, we initially utilized a Random Forest algorithm to determine match rates, which subsequently served as value parameters in our application of the knapsack optimization problem. Accompanied by predetermined constraints on the number of individuals and age range, we successfully curated an optimal group of ten participants for the speed dating cohort. Herein, we leveraged the knapsack algorithm to optimize for match rates, ultimately yielding a select group that met our established criteria.

This methodical approach allowed us to not only predict but also enhance the matchmaking process, culminating in a well-defined assemblage poised for speed dating. The employment of the knapsack problem in this context facilitated a focused optimization of potential matches, ensuring that the final selection aligns with the desired characteristics and constraints.



Algorithm 1 Random Forest Algorithm

for each tree in the forest **do**
    Randomly select a subset of samples (Bootstrap sampling)
    Build a decision tree:
    for each node **do**
        Randomly select a subset of features
        Find the best split point on these features
        Split the node into child nodes
    end for
    Repeat until max depth is reached or no further splits can be made
end for
To predict a new data instance:
for each tree **do**
    Produce a prediction
end for
Use majority vote (for classification) or average prediction (for regression) as the final prediction

2.3 Knapsack Dynamic Programming

The 0-1 Knapsack problem is a classical problem in combinatorial optimization. The problem models a scenario where one must optimally fill a knapsack with items of given weights and values, without exceeding the knapsack's capacity.[3][4]

2.3.1 Problem Definition

The formal definition of the 0-1 Knapsack problem is given by the following parameters:

- Let n be the number of items,
- Let w_i be the weight of item i,
- Let v_i be the value of item i,
- Let W be the maximum weight capacity of the knapsack,
- Let x_i be a binary variable that is 1 if item i is included in the knapsack and 0 otherwise.

2.3.2 Objective Function and Constraints

The objective of the 0-1 Knapsack problem is to maximize the total value V of the knapsack, which can be formulated as:

$$V = \sum_{i=1}^{n} v_i x_i \tag{1}$$

This objective function is subject to the following constraint, ensuring that the total weight of the selected items does not exceed the knapsack's capacity:

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

The variables x_i are binary, i.e., $x_i \in \{0, 1\}$.

2.3.3 Dynamic Programming Solution

A dynamic programming approach to solving the 0-1 Knapsack problem involves constructing a table K where each entry $K[i][w]$ represents the maximum value that can be achieved with the first i items considering the current weight limit w. The final entry $K[n][W]$ will then hold the maximum value possible for the knapsack capacity W.

```
function knapsack(values, weights, capacity):
    n = length(values)
    let K be a new array (n+1) by (capacity+1)

    for i from 0 to n:
        for w from 0 to capacity:
            if i == 0 or w == 0:
                K[i][w] = 0
            else if weights[i-1] <= w:
                K[i][w] = max(values[i-1] + K[i-1][w-weights[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][capacity]
```

2.4 Knapsack Application in Speed Dating

The algorithm presented in this report is an innovative approach to a matchmaking optimization problem, which is a specific application of the classic 0/1 Knapsack problem. It is designed to select an optimal combination of potential partners to maximize compatibility within given constraints.

2.4.1 Mathematical Formulation

The objective function to maximize the total match rate is given by:

$$\text{Maximize } Z = \sum_{i=1}^{n} v_i \times x_i$$

Subject to the constraints:

$$\sum_{i=1}^{n} x_i \leq N$$

$$\sum_{i=1}^{n} a_i \times x_i \leq A_{\max}$$

$$x_i \in \{0, 1\} \quad \text{for all } i$$

The recursive relation for dynamic programming is given by:

$$V[i, k, j] = \max\{V[i-1, k, j], V[i-1, k-1, j-a_i] + v_i\}$$

where a_i and v_i denote the age difference and match rate of the i-th partner, respectively, and x_i is the binary decision variable.

2.4.2 Problem Definition

Potential Partners: Each partner is characterized by two attributes:

- **Absolute values of Age Difference (Weight)**: The Absolute values of age difference between the partner and the reference individual, serving as the 'weight' in this context.
- **Match Rate (Value)**: A numerical score representing the compatibility or match rate with the reference individual.

Constraints:

- **Total Age Difference Limit (Bagweight)**: This is the maximum allowed cumulative age difference between all chosen partners.
- **Person Limit**: The maximum number of partners that can be selected.

2.4.3 Algorithm Description

The algorithm employs a 3D dynamic programming table dp, where $dp[i][k][j]$ stores a tuple containing the total match rate ($value$) and a list of indices representing the chosen partners, given the first i items, with a limit of k persons and a total age difference not exceeding j.

1. **Initialization**: A table dp of dimensions $(n+1) \times (person\_limit+1) \times (bagweight+1)$ is prepared. Each cell starts with a tuple $(0, [])$.
2. **Table Construction**: For each potential partner i, person limit k, and age difference limit j:
   - If $j < weight[i-1]$, the algorithm retains the value from the previous computation.
   - Otherwise, it evaluates:
     - **Excluding Partner** i: Takes the value from $dp[i-1][k][j]$.
     - **Including Partner** i: Updates the total value by adding the match rate of partner i to $dp[i-1][k-1][j-weight[i-1]]$ and adds the index $i-1$ to the selection list.
   - The optimal value between these two scenarios is stored in $dp[i][k][j]$.
3. **Solution Retrieval**: The maximum total match rate and the indices of the chosen partners are extracted from $dp[n][person\_limit][bagweight]$.

3 Structure of src

The implementation of 0-1 knapsack in the setting of speed dating, and the class RandomForestClassifier can be found in **source.py**. In the tools.py, we put the data preprocessing script for **speeddating.csv**, and generated **selected_columns.csv** using the script. Below is a description of the main functions:

The `RandomForestClassifier` is an ensemble machine learning model that operates by building a multitude of decision trees at training time and outputting the class that is the mode of the classes predicted by individual trees. Its key functions include `fit(X, y)`, which trains the forest of decision trees on the given input features X and target labels y, and `predict(X)`, which aggregates predictions from all the trees to determine the final class labels for the input samples X.

The `onePerEach` function solves a variant of the 0/1 Knapsack problem optimized for matchmaking. In this context, the objective is to select a subset of potential partners to maximize the sum of their match rates while adhering to two additional constraints. Firstly, the total age difference between the selected partners must not exceed a specified limit (bagweight). Secondly, the number of partners chosen must not exceed a set limit (person_limit). Each potential partner is represented as an item with two attributes: an 'age difference' that contributes to the total weight and a 'match rate' that adds value. The function identifies the optimal combination of partners that provides the highest sum of match rates without the total age difference exceeding the bagweight and without

selecting more partners than the specified person_limit. The function returns a tuple containing the maximum value that can be attained, the list of indices of chosen items (potential partners), and the list of values (match rates) of the chosen items.

Remark: The convergence of Random Forest can be found in the folder example in git.

4 A Description of What Tests are in Test

The tests can be found in **tester.ipynb**. For the tests, we firstly test that some essential functions like predict and predict_proba in Random Froest Classifier works properly, and make sure that the Random Forest Classifier indeed achieve satisfactory accuracy. We then test the implementation of 0-1 knapsack algorithm in the setting of speeding dating. Following this, we compared the total sum of match rates obtained from the knapsack model with the top 10 match rates. Additionally, we created a one-dimensional scatter plot to compare the positions of the points obtained from the knapsack model with those of the top 10 match rates.

5 Investigations into the Effectiveness of the Methods

5.1 Accuracy of Random Forest Classifier

Firstly, we tested our Random Forest Classifier. As the match rate predicted by the Random Forest Classifier is a key element to the application of knapsack in speed dating setting, we want to make sure it has high accuracy.
To achieve this, we then using train_test_split from from sklearn.model_selection to attain training and testing sets. After fitting the Random Forest Classifier to the training set, we tested its performance on the testing sets using the following set of metrics: Accuracy, Precision, Recall, and F1 score, we found the following results

Metric	Value
Accuracy	0.8190
Precision	0.7888
Recall	0.8190
F1 Score	0.7955

Table 1: Random Forest Classifier Evaluation Metrics

Figure 1 presents a confusion matrix resultant from a Random Forest classification model. The matrix offers a quantitative assessment of the model's performance, with the true class labels along the vertical axis and the predicted class labels along the horizontal axis. The top-left cell indicates the true negatives, with the model correctly identifying 162 instances as the negative class, which comprises 94% of the total negative predictions. Conversely, the bottom-right cell shows the true positives, where 10 instances were accurately predicted as the positive class, constituting 26% of the actual positive cases.

The off-diagonal cells denote instances of misclassification; the top-right cell represents false positives, with 10 instances erroneously classified as positive (6% of the negative class), and the bottom-left cell accounts for false negatives, where 28 instances were incorrectly labeled as negative (74% of the positive class). The matrix's diagonal reflects the model's predictive accuracy, with darker shades representing higher densities of correctly predicted instances. This visualization underscores the model's higher precision in predicting the negative class over the positive class.

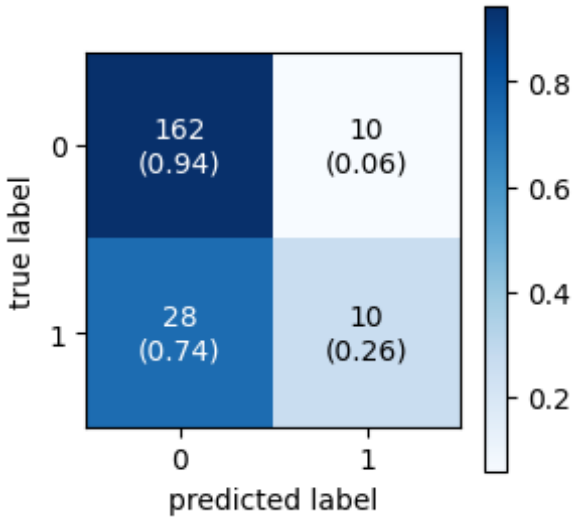


Figure 1: Confusion Matrix of Random Forest Classification

5.2 Convergence Analysis of Modified Random Forest

Figure 2 presents the convergence analysis of a modified Random Forest classifier. As the number of trees in the ensemble increases, the accuracy of the model initially shows a significant increase, indicating that the ensemble benefits from each additional decision tree. The rapid improvement in accuracy can be attributed to the diversity that each new tree brings to the ensemble, thereby enhancing the model's ability to capture various patterns within the data.

After the sharp rise, the accuracy reaches a plateau, suggesting that the model starts to converge. This plateau indicates that adding more trees beyond a certain number provides diminishing returns in terms of improving the model's accuracy. It becomes evident that the model has reached its optimal performance and further growth in the number of trees does not contribute significantly to the predictive power of the ensemble.

This behavior is typical in Random Forest classifiers, where after a certain point, the benefits of adding more trees taper off. The convergence point, in this case, seems to occur after approximately 20 trees, as the accuracy stabilizes around 0.9. The flat line following the convergence point suggests that the model has achieved a balance between learning the underlying patterns in the data and generalizing well to unseen data.

The convergence graph is crucial in determining the appropriate number of trees to use in practice, as it directly relates to the computational cost and the overall efficiency of

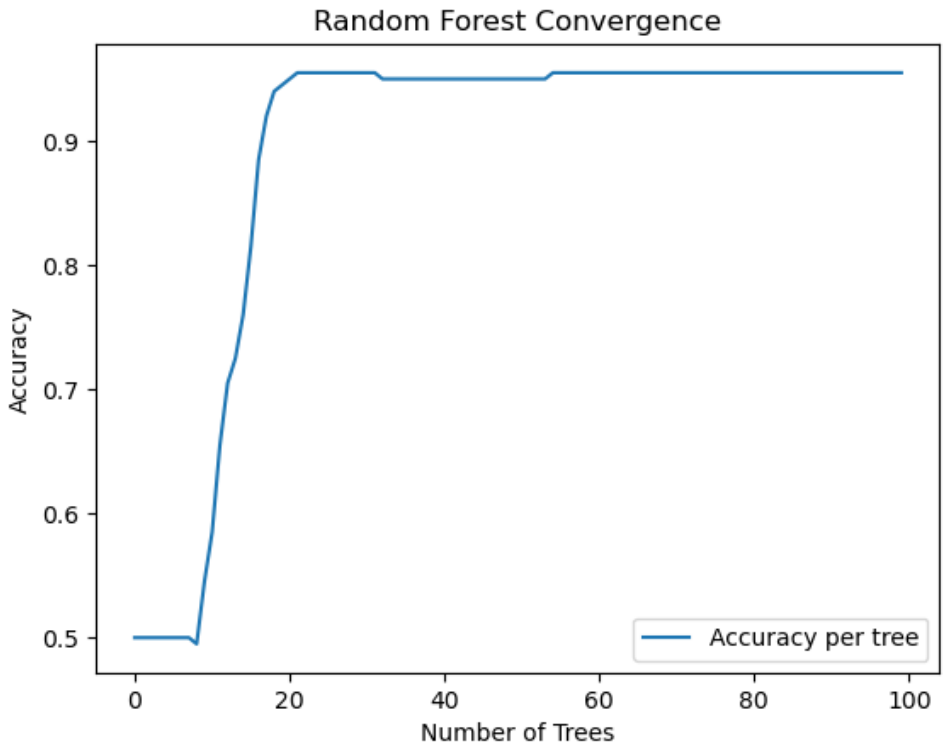


Figure 2: Convergence of the Modified Random Forest algorithm with the increase in the number of trees.

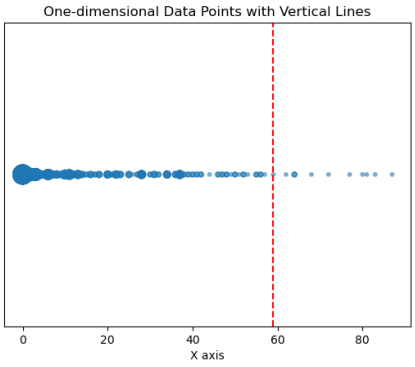
the model training process. In this modified Random Forest, the stabilization of accuracy suggests a robust model that generalizes well without overfitting, making it suitable for practical application with the chosen number of trees.

5.3 Effectiveness of Knapsack

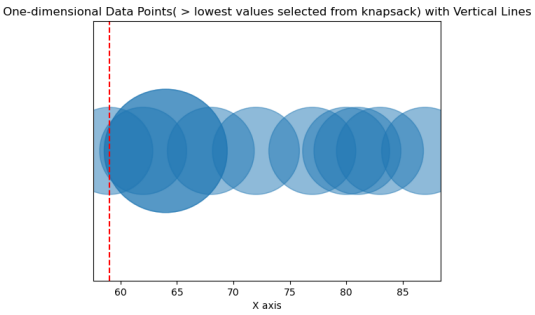
In our knapsack model, we imposed a limit of selecting a maximum of 10 individuals and introduced a total age difference constraint of 20. This choice was guided by insights from Emma's research [5], indicating that American males generally favored smaller age differences between themselves and potential partners, with an average preference of 1.65 years. To facilitate a more adaptable selection process while adhering to the restriction on the number of individuals, we made the decision to set the total age difference constraint to 20.

Based on the Figure 4, it becomes evident that the sum of match rates for the ten individuals selected by the knapsack algorithm is 735. In contrast, the sum of match rates for the top ten individuals with the highest rates in our dataset is 738. This observation suggests that the knapsack algorithm efficiently accomplished the task of finding an optimal solution based on match rates.

Subsequently, we obtained the ranking of the lowest match rate among the ten individuals selected by the knapsack algorithm from our Jupyter notebook, and it was ranked eleventh. This ranking demonstrates the efficiency of the knapsack algorithm in selecting the optimal compatibility scores.



(a) Scatter Plots of Data



(b) Scatter Plots of Data(values larger than lowest match rate by knapsack)

Figure 3: Scatter Plots of Data

```
[64, 77, 68, 64, 87, 59, 83, 81, 72, 80]
```

(a) Output match rates (in percentage) by knapsack

```
In [15]:  1  onePerEach(weight, value, 20, 10)

(735,
 [1, 9, 33, 52, 91, 138, 145, 150, 153, 180],
 [64, 77, 68, 64, 87, 59, 83, 81, 72, 80])

In [25]:  1  data = np.sort(value)[-10:]

In [26]:  1  sum(data)

738
```

(b) Image 4

Figure 4: Sum of match rates (in percentage) comparison

Furthermore, we can deduce from the one-dimensional plot that at the x=64 position, the circle is larger than at other positions. This implies that there are two data points with compatibility scores equal to 64. When calculating the count of data points around the circle's center, we also find that 59 is the eleventh-ranked compatibility score. Therefore, it can be concluded that the knapsack algorithm efficiently selected the top ten compatibility scores.

However, it's important to note that the knapsack algorithm exhibits a limitation in terms of its time complexity, which led to its application only on the x-test data.

References

[1] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[2] ——, *Classification and regression trees*. Routledge, 2017.

[3] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: a survey," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975.

[4] S. Martello and P. Toth, "Algorithms for knapsack problems," *North-Holland Mathematics Studies*, vol. 132, pp. 213–257, 1987.

[5] E. Otta, R. da Silva Queiroz, L. de Sousa Campos, M. Weronika Dowbor da Silva, and M. T. Silveira, "Age differences between spouses in a brazilian marriage sample," *Evolution and Human Behavior*, vol. 20, no. 2, pp. 99–103, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1090513898000415