CS 5112 Final Project
Exploring Matching Approaches for Online Dating

Cornell Tech

Kristjan Ari Tomasson
kt476@cornell.edu

Eesha Khanna
ek542@cornell.edu

Anjali Vyas
av379@cornell.edu

Andrew Yan
zy434@cornell.edu

December 18, 2021

1 Introduction

Around 20% of new romantic relationships in the USA begin online. People browse dating apps to swipe through different profiles, looking for their perfect match. The success rate of this process is highly affected by which profiles the platform displays to the user. It is well known that dating platforms collect information about users and use algorithms to try and predict likely matches. While the implementations of these algorithms are generally not public information, it is believed that they usually involve common techniques of recommendation systems, such as collaborative filtering. In this study, we explore a different approach to matching users, using various types of clustering-based methods as well as a nearest neighbors approach. It is assumed that users tend to prefer partners that have similar profiles to themselves. The goal of the project is to assess different algorithms and approaches to split users into clusters based on their profiles, where similar profiles are placed in the same group. Then, matching suggestions are formed within these groups based on users' sex and orientation.

1.1 Data

The dataset used is sourced from Kaggle. It contains roughly 50,000 user profiles from a dating app called OKCupid. OKCupid has more robust user profiles compared to most other dating apps and hence the dataset was suitable for the purpose of this project. It contains more than 20 features about each user which were analyzed to create groups of similar user profiles. The data was preprocessed before any algorithms were applied, where irrelevant features were removed and similar features were combined. Features that were subject to matching purposes after grouping were then put aside, e.g. sex, orientation and location. All responses to open-ended essay questions were also removed as sentiment analysis is out of the scope of this project. After preprocessing, the dataset consisted of 10,000 profiles with the features listed in table 4.

age	body type	diet	drinks	drugs	education	ethnicity	zodiac signs
height	job	offsprings	pets	religion	smokes	languages	

Table 1: User features

To allow for distance calculations between user instances, the categorical columns were one-hot-encoded and the numerical ones were scaled down to an interval between 0 and 1. Note that certain algorithms did not require the scaling and one-hot-encoding preprocessing steps. Any exceptions to preprocessing are described in section 2, under each algorithm's sub-section.

While age and height are the only numerical features, 'drinks' and 'body type' were converted to be ordinal features. While they seem like categorical variables at first glance, they have order within their values and can be placed on a scale of 0 - 5 (for instance, 0 = user does not drink and 5 = user drinks a lot). Therefore it was concluded that it would be beneficial for the objective to make use of this ordinality and treat these as numerical features.

1.2 Algorithms

Various clustering algorithms along with K-Nearest Neighbors were applied to split the user database into groups of similar users. The clustering methods observed were K-means, hierarchical clustering, K-prototypes, and DB-Scan. We experimented with different distances metrics on some of these algorithms.

2 Process

The algorithms listed above were run on the dataset and parameter tuning was then conducted for each of them to try to optimize their performance. External libraries were used to access implementations of the algorithms. The data consists of 10,000 user instances and we hoped to find between 100 and 200 possible matches for each user. This is achieved by splitting the user base into 50 groups, assuming the groups are roughly the same size. Some of the algorithms require the number of clusters to be given a priori; in these cases, the number 50 is used. After adapting the algorithms to the data, their performance was then evaluated. Below, each algorithm and its application to the data are analysed.

2.1 Evaluation methods

The dataset contains various attributes about individual users but there is no information of which users actually matched in real life. Therefore, the ground truth of the dataset is not apparent. In order to overcome this limitation, we used two different methods to evaluate performance: visual inspection as well as calculating accuracy against a small subset of the data which was manually labeled.

2.1.1 Visual Inspections

Since our data is high dimensional and it is not possible to view the plotted data points in such a high dimension, we decided to use a dimensionality reduction technique called UMAP - Uniform Manifold Approximation and Projection - to represent a 2-dimensional projection of the data in order for us to view the clusters visually [1]. Note that these visualizations were not used to formally evaluate the algorithms as information is lost in this dimensionality reduction. They are solely used to get a visual sense of clustering results and to perform a gut check that the algorithm is performing reasonably well. Since we set the number of clusters to be 50 and it is hard to visualize 50 different colors, we decided to only visualize a smaller number of clusters (8) with the assumption that the algorithm would perform similarly on a smaller number of clusters and allow us to still perform that initial check. These visualizations do not by any means evaluate the quality of the clusterings.

2.1.2 Calculating accuracy against hand-labeled data

In order to evaluate the clusters created by each algorithm, we took the approach of manually labeling 200 pairs of users. The underlying assumption behind our overall approach is that users with similar attributes would be a good match. Therefore the hand-labeled users are split into 100 pairs, 50 of those are especially similar users that should be placed in the same cluster while the other 50 pairs are dissimilar users that should not be placed in the same cluster. The accuracy of the algorithms is then evaluated by observing the clustering labels of those users. An algorithm that performs well groups similar pairs together while it places dissimilar pairs into separate groups.

$$\text{Similar accuracy score (\%)} = \frac{\text{number of similar pairs that the algorithm placed in the same clusters}\times 100}{\text{number of hand-labeled similar pairs}}$$

$$\text{Dissimilar accuracy score (\%)} = \frac{\text{number of dissimilar pairs that the algorithm placed in different clusters}\times 100}{\text{number of hand-labeled dissimilar pairs}}$$

2.2 Weighting features

In the context of matching people based on their similarity, some features are more relevant than others. The clustering algorithms consider 18 features of each user and they are all considered equally important in the dataset. However, some of them are more applicable for the purpose of this project. In order to account for this we classified the features into ranked groups depending on how likely they are to affect people's opinion on a romantic partner. Age, education, smoking habits, and drug consumption were weighted the highest; height, ethnicity, body type, drinking habits, languages spoken, whether a user likes pets, and whether a user has/wants kids were weighted the second highest; diet, job, and religion were weighted second lowest, while a user's belief in zodiac signs was weighted the lowest.

The weights were introduced in the clustering process for certain algorithms by giving each feature the corresponding weight from above when calculating distance between user instances. The weights were implicitly considered when the data was hand labeled and we were interested in observing how adding weights would affect the clustering quality in the experiments below.

2.3 Algorithms

In this section each algorithm is explained and how they were applied to the dataset. There is one section for each algorithm, explaining how the experiment was set up before the outcome is analyzed.

2.3.1 Hierarchical clustering

Hierarchical clustering methods seek to build a hierarchy of clusters. In this project, experiments were run with agglomerative clustering, a bottom-up version of hierarchical clustering. Each user starts as its own cluster, and then the most similar clusters are combined until they are all part of the same cluster. A dendrogram represents the clustering process at every step on the way and can be used to determine how many clusters are the most suitable for the dataset. We made use of the sci-kit learn implementation of this algorithm[4].

Experiments Three parameters of the algorithm were analysed: the final number of clusters, the linkage which determines which distance is used between sets of observations, and the distance metric used to calculate the distance between two users. We used two distances metrics - L1 distance and Euclidean distance. For each of these, we also experimented with a weighted version, where each feature was given a weight based on the description in section 2.2. The weighted distance metrics are described below:

- Weighted L1 distance (between users i and j) =

$$d_{l1}(i,j) = \sum_{a \in S} w_a * |v_{i,a} - v_{j,a}|$$

  where w_a represents the weight for each attribute a, in the set of post-one-hot-encoded attributes S, and v_{ia} represents the value in the above input data matrix at user i, and attribute a.

- Weighted Euclidean distance (between users i and j) =

$$d_{euclidean}(i,j) = \sqrt{\sum_{a \in S} (w_a * v_{i,a} - w_a * v_{j,a})^2}$$

To analyse the number of clusters, a **dendrogram** was generated by running agglomerative clustering on the 10,000 users (Figure 1). Each individual x-axis unit represents a user in the data set, while the y-axis represents the cost of each step of clustering agglomeration. The diagram can then be analysed to determine the optimal number of clusters for the dataset. As discussed above, the objective is to split the users into 50 groups, and for the purpose of consistency in accuracy comparison, we decided to use 50 clusters for the experiment.

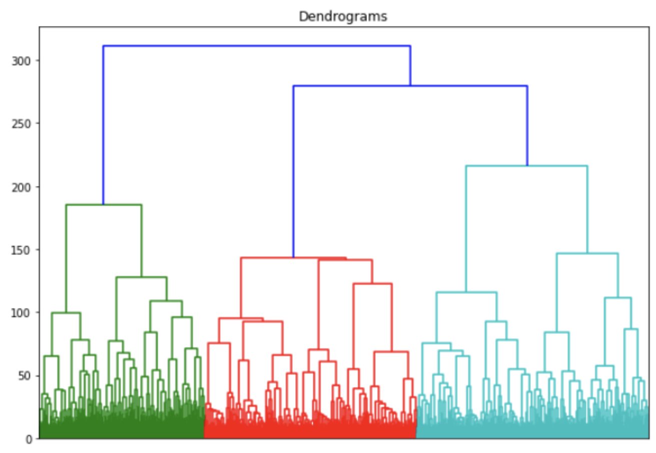

Figure 1: Dendrogram for hierarchical agglomerative clustering

The four linkage metrics used are listed below:

- **Average** linkage uses the average of the distances of each observation of the two sets.
- **Ward** linkage minimizes the variance of the clusters being merged.
- **Complete or maximum** linkage uses the maximum distances between all observations of the two sets.
- **Single** linkage uses the minimum of the distances between all observations of the two sets.

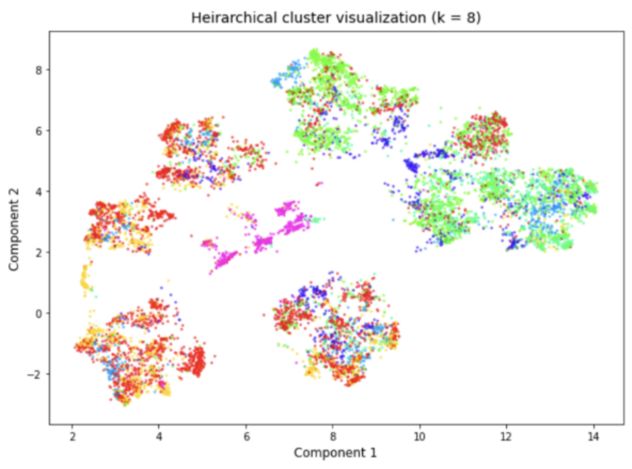

Figure 2: Visualization of hierarchical clustering

Insights and discussions Before any parameter tuning was conducted, the simple agglomerative clustering method was run on the dataset and the results were visualised using the UMAP embedding for k=8 (Figure 3). The figure shows the clustering of the algorithm which seemed to be fairly fit for the task at first glance. The clusters are relatively clearly separated and evenly sized.

After applying the algorithm to the users the parameters were tuned to find the best performing combination. The table below shows the evaluation results for the agglomerative methods.

	Similar accuracy	Dissimilar accuracy	Distribution standard deviation
Euclidean distance w average linkage	92%	96%	905.63
Euclidean distance w single linkage	100%	21.57%	1392.71
Euclidean distance w complete linkage	72%	100%	317.10
L1 distance w average linkage	96%	82.35%	1110.98
L1 distance w single linkage	100%	15.69%	1392.71
L1 distance w complete linkage	56%	100%	296.10
Euclidean distance w ward linkage	**68%**	**98%**	**94.1**

Table 2: Hierarchical Clustering results with different methods

As we can see from the table, the choice of linkage method seems to impact accuracy more than the choice of the distance function. The average linkage method performed the best among others when considering accuracy for the hand labels. We observe that single linkage performs best for the similar accuracy while complete linkage outperforms all others under the dissimilar accuracy. This is not surprising as the single method aims for most users to be in clusters with users that are fairly similar to them on average while the complete one considers the maximum distance and hence it does not place users that are very different in the same cluster.

For the purpose of this project, it is more important that users are generally placed in groups with similar people than making sure no two dissimilar users are in the same group. This is because users will be able to review all users within their group and can simply reject the ones they do not like when selecting matches. However, the single method is too aggressive in placing similar users in the same cluster and seems to put most users into the same cluster to make sure similar users are grouped together. The average method strikes the right balance and seems to perform the best holistically as it considers both factors - grouping similar users while separating dissimilar users. The ward linkage method provides a reasonable descent but not top of the line accuracy score.

Cardinality and accuracy trade-off A distinguishing characteristic of hierarchical clustering from other algorithms is that it tends to place an uneven number of people within a cluster. This might be beneficial in providing a more accurate cluster and having lower cost/variance than even cardinality clustering algorithms such as K-means. However, the down side is that if certain subset of users is very different from other sets of users, the 'different' users would get placed in very small clusters, and would therefore have very few matches.

In our case, while the average linkage method and the euclidean distance function seems to perform the best in terms of hand labeling accuracy, it performs poorly when it comes to generating somewhat evenly sized clusters. As we can see from the figure comparison below, the best performing (average linkage) contains two clusters with the majority of the people, leading to an extreme unbalanced distribution, whereas the ward method produces a more even distribution but with a lower accuracy score. This trade-off is important and should be considered in actual practice. Having the majority of the users in the same cluster provides no information for the objective of the project. Hence, we believe that the ward linkage method is more appropriate for our use case - even though the accuracy is lower, it splits the user database into relatively similar sized clusters which can be used for the matching objective.

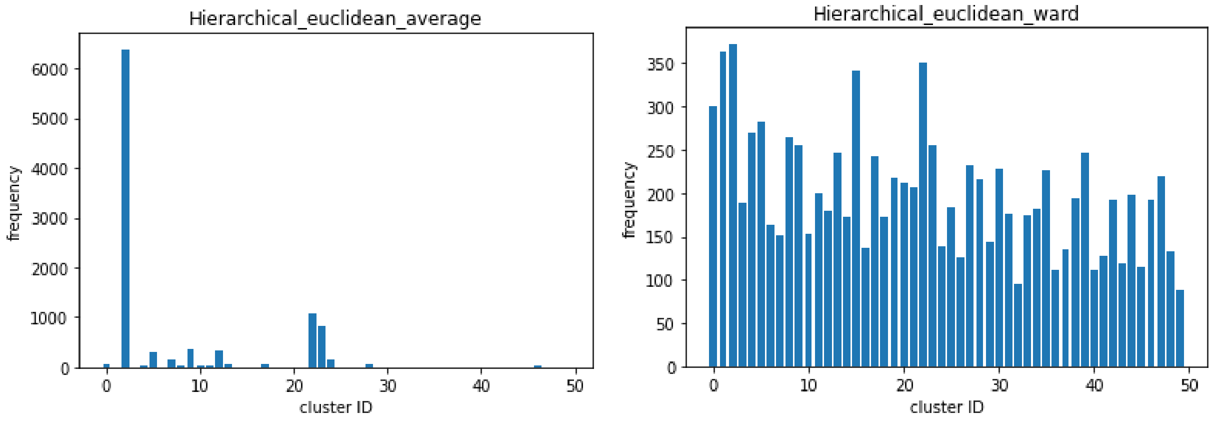

Figure 3: Visualization of hierarchical clustering

2.3.2 K-Means

We also experimented with the K-means algorithm as a baseline comparison with the clustering results from hierarchical clustering. K-means clustering is a relatively simple clustering algorithm that assigns data instances to k clusters and iteratively improves the clustering quality by moving cluster centroids to the mean of its data points and then re-allocating data points to the closest cluster.

Experiments The main parameter of the algorithm is the number of clusters. However, this parameter is pre-decided to be 50 as previously mentioned. The algorithm can however be very dependent on the initial choice of cluster centroids which are generally chosen randomly. We used scikit-learn's implementation of the K-means algorithm [4]. This implementation attempts to overcome the limitation of a random initialization by running the algorithm 10 times with different initial centroids each time, and outputting the best clustering result out of the 10 runs.

For the K-means algorithm, the euclidean distance measure is generally used and there was no change made to that rule in this application.

Insights and Discussions As with hierarchical clustering, the algorithm seemed to be applicable to the task. Figure 4 shows a visualization of 8 clusters generated by the algorithm. Again, they seem to be fairly well separated and the difference in cluster sizes is visually insignificant. Note that this is only a visual inspection, and not a robust evaluation of the algorithm however. The accuracy scores for the algorithm with 50 clusters are analysed in the final results section below.

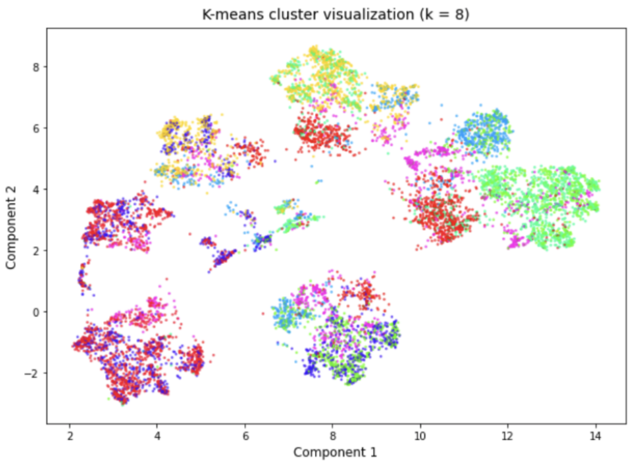

Figure 4: Visualization of K-Means clustering

2.3.3 K-Prototypes

K-prototypes is an algorithm which combines K-means and K-modes in order to efficiently cluster mixed data types (numerical and categorical variables). It is a clustering method based on partitioning. It measures distance between numerical features using Euclidean distance (like K-means), while distance between categorical features is measured using the number of matching categories using a matching dissimilarity measure [2]. Instead of using the mean as the cluster centroid (as in K-means), the K-prototypes algorithm replaces the means of clusters with modes for categorical features, and uses a frequency-based method to update modes in the clustering process to minimise the clustering cost function.

We used Nelis J. de Vos's implementation of the K-prototypes algorithm in our study [3]. This implementation has the preprocessing steps built in, and therefore we did not pass in the standardized, one-hot-encoded dataset, but rather just the original cleaned version of the dataset into the algorithm.

Experiments The first step was to find the optimal number of clusters for our dataset. While we had an intuition for the number of clusters based on our general understanding of how many users should be in a cluster on average (as detailed in the beginning of section 2), we wanted to confirm this assumption. We plotted the clustering cost as a function of the number of clusters and used the elbow method to pick the optimal number of clusters.

While the initialization for the means (for the numerical features) is randomized, like in a naive K-means implementation, we experimented with two different methods for initializing the modes of the clusters (for the categorical features). The first method simply selects the first k distinct objects as the initial K-modes, while the second method calculates the average density of the data based on the frequency of attribute values, and assigns the initial modes based on the distance between points as well as the density of points. The second method was proposed as an improvement to the original method by Cao et al.

Insights and Discussions Based on the results from the elbow method plot, we found that our initial assumption about using 50 clusters was partially confirmed (Figure 5). While the 'elbow' of the line seems to be around k = 40 as the marginal cost reduction of adding more clusters goes down beyond this point, we decided to stick with the initial assumption of 50 clusters as it still seems like a reasonable assumption based on this plot.

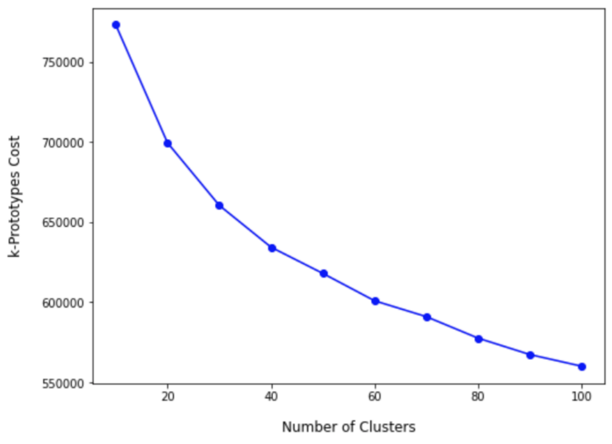

Figure 5: Elbow method for K-Prototypes

While we were still tuning the parameters, we used the UMAP embedding to visualize the results for k=8 clusters (Figure 6). We found that the clusters looked more spread out visually as compared to the UMAP visualizations for K-means and hierarchical clustering. Initially, we thought that this was because K-prototypes was not performing as well, but further investigation revealed that K-prototypes was performing comparably well, but the 2 components that are being visualized are not capturing the variance of the categorical variables that K-prototypes was relying more to cluster the data on. This finding reveals another limitation of the UMAP visualization and reinforces the need for a more robust evaluation method.

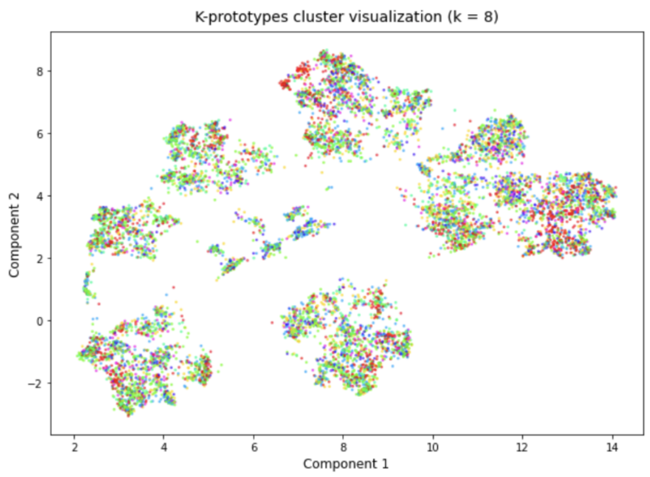

Figure 6: Visualization for K-Prototypes clustering

Further, as we experimented with the different initialization methods, we found that the second method of initialization, while relies on data density and distances to pick the initial cluster centers yielded slightly better results from an accuracy score perspective. Given that this method of initialization makes more of an 'informed guess' for the initial cluster centers, this finding is in line with what we expected. We decided to use the second method of initialization to evaluate the performance of the algorithm. The similar accuracy score was 32%, while the dissimilar accuracy score was 100%.

The K-prototypes algorithm by default assumes that all features have the same weight. Most features of our dataset only have 1 column associated with them. The only exception is the 'languages spoken' feature. We decided to break this up into one column per language, with the values being yes/no based on whether the user speaks that language or not. We thought this would be better because if this information was represented in one column, each user would have a list of languages associated with them and the algorithms would treat each new combination of languages spoken by users as a 'new category.' With this, if 2 users had 2 languages in common, they would not be considered similar in any manner from the language perspective unless those were the only languages they had in common.

However, this representation did not seem to work well with K-prototypes. As we were inspecting the clustering results manually, it seemed that several of the cluster results contained users with the same value for binary categorical features, but varying values for features with several possible categories. Since the K-prototypes algorithm does not one-hot-encode the categorical variables, we believe that the 'languages spoken' feature (with 10 columns) was over-represented in the input and was therefore skewing the results. Some of the categorical features like 'education' and 'job' had more categories but since all features were weighted the same, the cost function indicated a 'good' performance if the algorithm ensured all language categories had the same values, but arguably more important features like education and job still varied quite significantly within a cluster. We realized that weighting columns differently would allow us to overcome this, but found that there was no streamlined way of doing this with the implementation that we were using. Therefore, in an attempt to get around the limitation of weighting features, we decided to drop a few columns for languages that were very rare (spoken by less than 50 users). We found that the similar accuracy score went up to 38%, i.e. improved by 6%, after making this change, and the dissimilar accuracy score stayed the same (100%).

2.3.4 DB-Scan

DB-Scan is a density-based clustering algorithm that groups points that are closely packed together while marking points in regions of low density as outliers. The most important parameter for the algorithm is epsilon, the maximum distance between neighbor samples. Apart from that, we also tuned other parameters like the distance metric and the minimum number of samples in each cluster.

The algorithm has some advantages over the ones listed above. It does not require a set number of clusters specified a priori and it can find arbitrarily shaped clusters. However, these advantages come with a cost as the algorithm struggles with clusters with high variance in densities and it is highly dependent on the chosen epsilon value, which can be hard to determine in high dimensional data - especially if the data is not fully understood. We made use of the sci-kit learn implementation of DB-Scan in our investigation [4].

Experiments Three distance measures were used with DB-Scan: Euclidean, Canberra, and Cosine. For each of those metrics the optimal epsilon value was determined by plotting the minimum distance from each point to its neighbours and using a knee locator - which finds the point of maximum curvature - on that result to identify the optimal distance threshold. Using that value, the points on the left of the point of maximum curvature should be core points in the clustering. Then, the parameter was tuned further by trying out 8-10 values of the same scale as the one identified with the method above. The values for each metric are displayed in Figure 10.

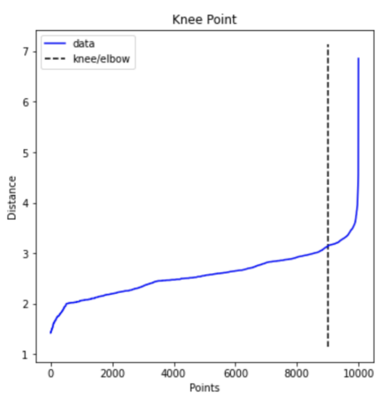

Figure 7: Euclidean

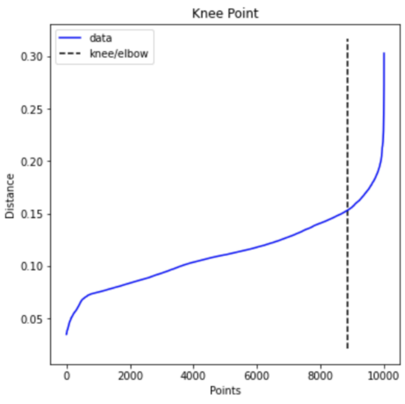

Figure 8: Cosine

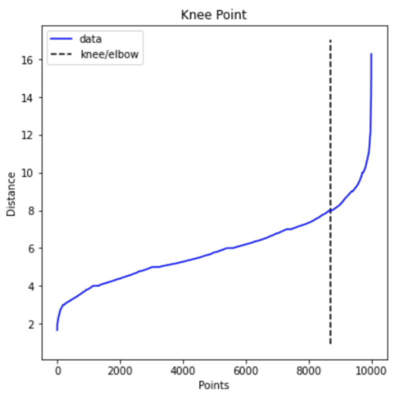

Figure 9: Canberra

Figure 10: Elbow methods for DB-Scan parameter tuning

For each of these values, the Silhouette score, which measures how similar points are to points within their own cluster, was calculated and the clustering was visualized with UMAP encodings.

Insights and Discussions The experiments with DB-Scan did not deliver meaningful results. The Silhouette score was acceptable in some cases, around 0.5, but these were all situations where the data was split into one huge cluster and the rest of the data points were outliers. For all the clustering results but one, 9,000 out of 10,000 samples were either in the big cluster or marked as outliers. The rest of the samples were then distributed in hundreds of small clusters, containing 1-10 users. In the one exception, the two big groups contained 8,000 of the users while the rest was split into 500 small clusters.

Figure 11 demonstrates how the DB-Scan clusters were typically structured. Almost all of the users were part of the large cluster (red) while some users were paired in very small clusters (colored dots).

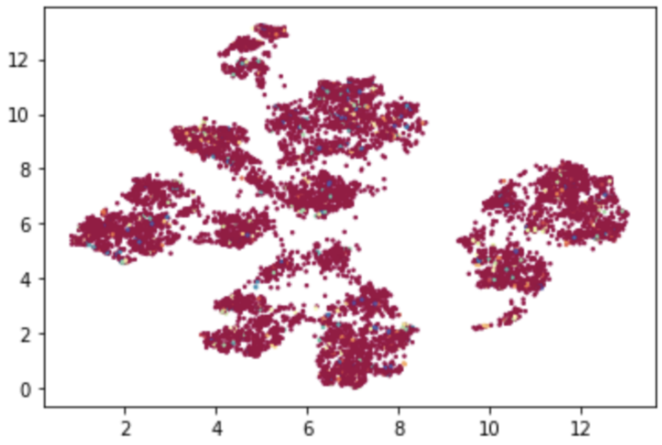

Figure 11: Visualization for DB-Scan clustering

These results suggest that the value of epsilon was not appropriate in any of the cases. The epsilon value, representing the minimum distance for instances to be considered neighbors, was in all cases too small or too large. In the former case, the majority of the points are not within epsilon distance of other users and are thus classified as noise by the algorithm. In the latter case, the majority of the users fall in the same cluster as the distance threshold fails to differ between users. They are all within epsilon distance of other core points. The most interesting result was for the cosine distance iteration, when we experimented with the smallest angle, 0.05. As this was the most promising result, further experiments were conducted in that area, with the following values of epsilon.

0.01	0.02	0.03	0.04	0.05	0.06	0.06	0.08

Table 3: Cosine values for DB-Scan experiment

For the lowest value, almost all users were classified as noise but as the value of epsilon was increased, more users moved into the one large cluster. Overall however, the results were similar to what we found earlier, where most of the users were either in the same larger clusters or were considered outliers, while the remaining users were split into hundreds of small clusters. The most noticeable change with different epsilon values was that users were moved from being outliers into the big cluster.

The objective of this project is to split the user base into smaller subgroups of potential matches. Unfortunately, these clusters are not helpful for this objective as the majority of the users are in the same cluster, thereby rendering the clustering step to be useless. When all users are in the same cluster there is no advantage in matching users from that cluster over matching them randomly.

As discussed above, it can be a difficult task to determine the optimal value of epsilon for high dimensional data and the algorithm struggles with clusters of varying densities. Both factors are certainly affecting the results here. While the algorithm considers under 20 features for each user, the dimensionality is increases significantly when categorical columns are one hot encoded and as a result, this clustering problem does not seem to be a good fit for the DB-Scan algorithm. This result is highlighted by the fact that we were not able to find an appropriate epsilon value even after various iterative experiments. We observed that while some users have exactly the same profiles, there are several other users that may have relatively similar profiles and could still be a match. The impact of using a density-based algorithm on our problem is best illustrated as follows: When epsilon is small, the algorithm manages to group the users that are very similar and marks other users as noise. On the other hand, when epsilon is larger, the algorithm manages to group users that are less similar and separate them from the majority, while the majority is all just put in the same cluster.

Overall, this experiment failed to provide useful clusters for the objective and was therefore not evaluated with the hand labeled data. A hierarchical version of DB-Scan could be experimented with to try to provide meaningful clusters for this dataset, such as HDBSCAN. Each layer would have a separate value of epsilon and in this case, that would mean the big cluster would be split up recursively again with a different value of epsilon. However, the objective of this project is not reliant on using DB-Scan and hence other clustering methods will be utilized to achieve the goal.

2.3.5 K-Nearest Neighbors

The K-Nearest Neighbors algorithm (KNN) finds the K-Nearest Neighbors for each input value provided to it. The results are often heavily dependent on the distance metric used to determine what makes two points close or far apart. For our investigation, we used the sci-kit learn implementation of the KNN algorithm [4].

Experiments We made use of the Gower distance metric to compute distances between each pair of individuals in our sample (using this PyPI implementation by Michael Yan[5]). This is because our dataset had a mix of categorical as well as numerical features and the Gower distance metric can be used in this case unlike many other metrics such as Euclidean, Manhattan and Hamming distance.

We decided to set the k-value to be 200 since we were ultimately hoping to match each person to around 100-200 others based on their similarity.

As mentioned before, we experimented with two versions of this algorithm - one where the features were unweighted and one where they were according to our understanding of what people tend to find more important when they are looking for someone to date.

Insights and Discussion As a sanity check measure, we used the UMAP embedding to informally visualize only a subset of the results of the algorithm - 200 nearest neighbors for 5 randomly chosen people in our dataset. Since the algorithm returned 200 neighbors for each point, coloring all of them would have resulted in a visual representation that is too cluttered and not useful. The representations are shown in the figures below.

We found that for the most part, each member in a group of neighbors was in close proximity to a significant number of other members on the plane of the first two UMAP components. Interestingly, we could see separated sub-groups within certain groups of neighbors that were more similar to one another. For instance, looking at the group of 200 neighbors colored in light purple we can discern 3-4 visually separated subgroups that likely have members who are more similar to each other than those in other subgroups.

The KNN results are further evaluated in the result section below.

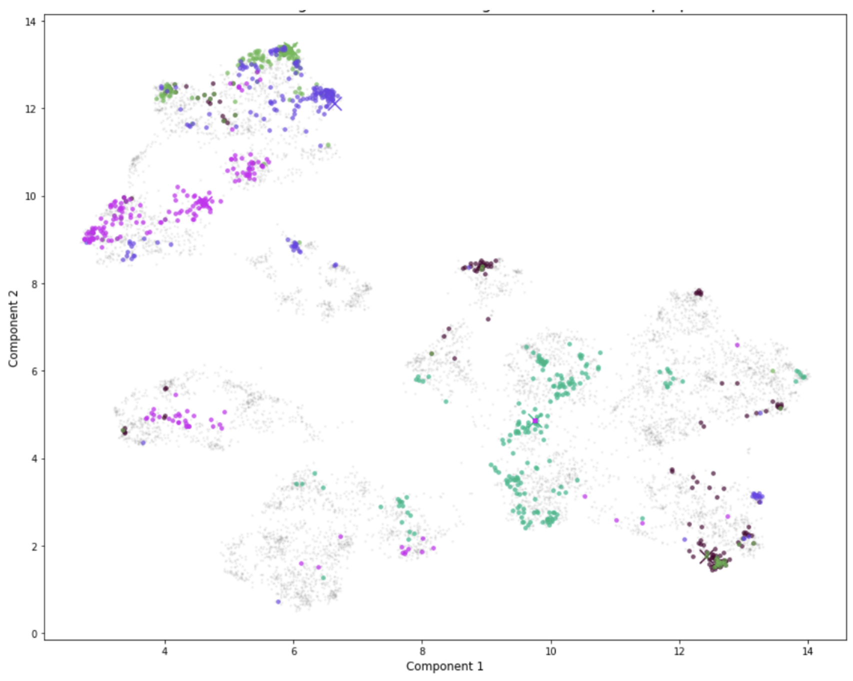

Figure 12: 200 nearest neighbors found for 5 randomly selected people using unweighted features

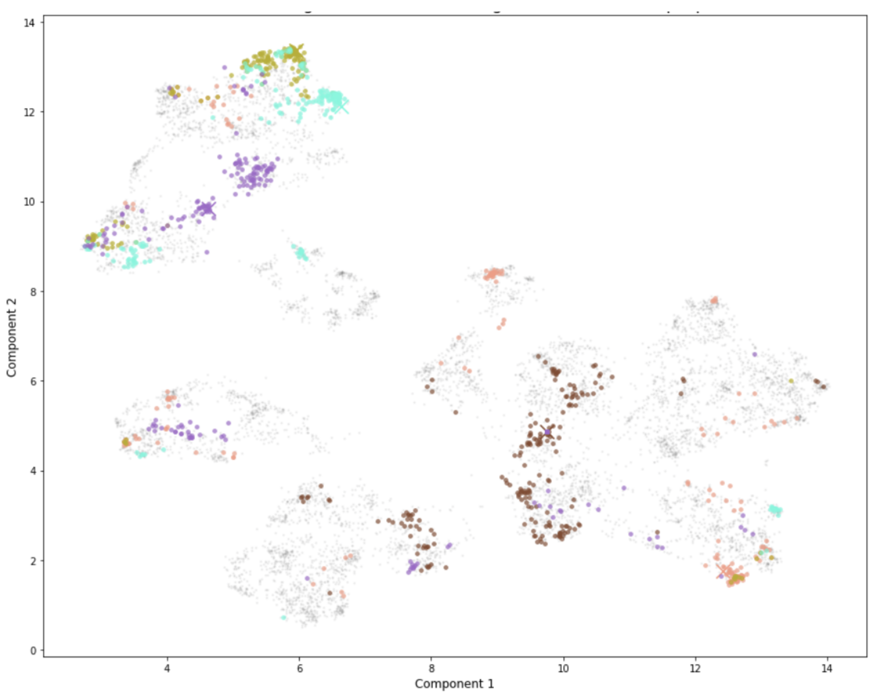

Figure 13: 200 nearest neighbors found for 5 randomly selected people using weighted features

3 Results

The table below shows the best results for all 4 algorithms that yielded clusters. The metrics shown in the table include similarity accuracy, dissimilarity accuracy, average empirical run time, as well as the standard deviation of the number of users per cluster. Since the average number of users per cluster is the same for all algorithms (200), we decided to include the standard deviation as a measure of the cluster balance. Low standard deviation indicates that the clusters are roughly the same size.

Amongst the clustering algorithms, we found that the weighted version of hierarchical clustering with the Euclidean distance measure and Ward linkage performed the best in terms of optimizing for accuracy, empirical run-time, as well as cluster distributions. We found that the K-prototypes algorithm took significantly longer to run. While the theoretical run-time for K-prototypes should not be too different from K-means, we suspect that K-prototypes took longer to run as the implementation is not as optimized as implementations for other algorithms in professional packages like scikit-learn [4]. It is important to note that the performance of weighted K-means was comparable to the best instance of hierarchical clustering and the accuracy only differed by 8%.

Overall, the weighted version of the KNN algorithm performed the best in terms of maximizing accuracy and minimizing standard deviation (since we only chose 200 'most similar' neighbors per user). However, KNN took a significantly longer time at 188 seconds. One could also argue that while KNN's approach of calculating exactly 200 'similar people' per user is more even and efficient, there could be cases where a user misses out on several matches because the hard cut-off at 200 prevents the algorithm from suggesting several other users that were almost equally similar to that user. On the other hand, it could also be the case that a user maybe only have a handful of similar users to themselves, but the algorithm forcibly suggests several additional not so similar users as potential matches. This limitation is exactly what hierarchical clustering overcomes.

Algorithm	Similarity Accuracy	Dissimilarity Accuracy	Approximate Empirical Time (s)	STD for number of users per cluster
K-means (unweighted)	30%	100%	10.4	54.3
K-means (weighted)	62%	98%	10.4	91.90
Hierarchical (unweighted, euclidean w ward)	28%	98%	13	68.5
Hierarchical (weighted, euclidean w ward)	68%	100%	13	94.01
Hierarchical (weighted, euclidean w average)	92%	96%	12.4	905.63
K-prototypes (unweighted)	32%	100%	1040	85
K-prototypes (fewer languages)	38%	100%	1040	95
KNN (unweighted)	66%	100%	188	0
KNN (weighted)	76%	100%	188	0

Table 4: Comparing the accuracy, empirical run times and number of people per cluster for the algorithms we explored

3.1 Matching users

After evaluating the performance of the algorithms for grouping people, the final task was to match users based on their sex and orientation. We wrote a simple algorithm that uses a user's sex and orientation to match them to relevant users within their cluster and ran this on the groupings from KNN as well as best instance of hierarchical clustering.

With the KNN results we could match each person to 80 others on average when the number of neighbors was set to 200 (with the weighted as well as unweighted version). The standard deviation of the number of matches each person got was around 33. As can be seen in the histogram (Figure 14), the vast majority of people got at least 60 matches but around $7 - 8\%$ of people (700-800) got less than 25 matches. Around 2% (186) of the people got less than 2 matches. Upon further investigation we found that all of the people who got 2 or fewer matches were gay. The vast majority of individuals in our dataset are straight. As a result, the nearest neighbors of gay and bisexual people are more likely to be mostly straight. Thus during the matching process, most of their neighbors will not be viable. To make our KNN algorithm more fair for people of all orientations we would have to reduce the constraints that make a person with a minority orientation similar to another person with that orientation.

For hierarchical clustering, we found that on average, each user got approximately 99 matches. The standard deviation of the number of matches each person get was around 57, which implies a huge variance in the number of matches. The minimum number of matches was found to be 1, while the maximum was found to be 277. The histogram (Figure 15) shows that the number of matches varies significantly more compared to the results from KNN. Only 8.76% of the users got less than 25 matches, and only two users got just 1 match. Approximately 42% of the users got more than 100 matches, and 6% of the users got more than 200 matches. A comparison with the KNN results reiterates the fact that these two approaches present a trade-off when it comes to maximizing cumulative utility (hierarchical clustering) versus minimizing the worst-case scenario (KNN).

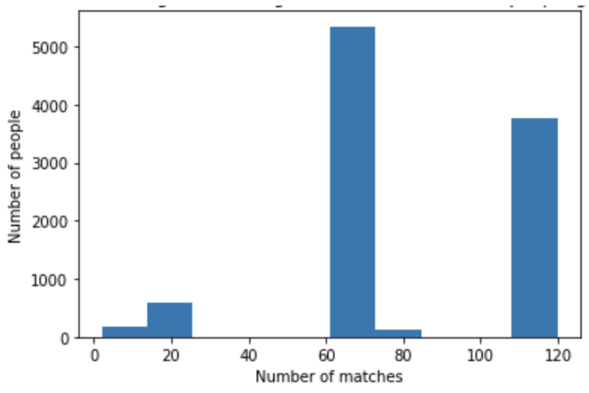

Figure 14: KNN - Histogram for number of matches per user

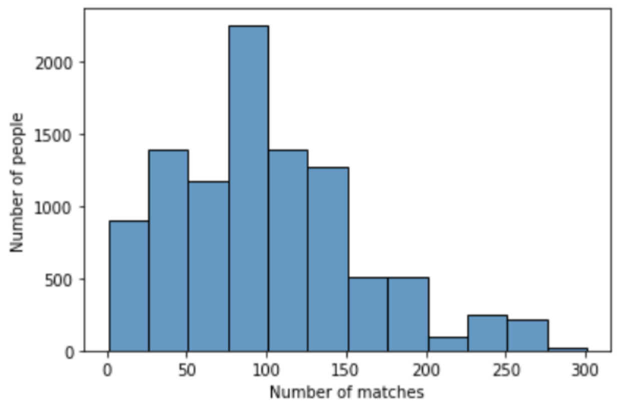

Figure 15: Hierarchical clustering - Histogram for number of matches per user

4 Conclusion

Overall, we found that our experiments were successful in matching dating app users using clustering and nearest neighbors techniques. As discussed above, both hierarchical clustering and KNN yield favorable results, and while KNN achieves a slightly higher accuracy, it takes longer to run. Both algorithms have their advantages when it comes to the distribution of clusters/groupings, and the decision to use one algorithm over the other would really depend on a dating app's priorities. If we were to deploy this matching technique in a dating app, we would probably choose to use hierarchical clustering over KNN so as to optimize for speed, but if a particular app is focused more on maximizing accuracy or ensuring that various users are recommended somewhat even number of matches, they may want to use the KNN approach.

References

1. McInnes L.: UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. 2017 - 2021. https://github.com/lmcinnes/umap.

2. Huang, Z.: Clustering large data sets with mixed numeric and categorical values, Proceedings of the First Pacific Asia Knowledge Discovery and Data Mining Conference, Singapore, pp. 21-34, 1997.

3. de Vos Nelis J.: kmodes categorical clustering library. 2015 - 2021. https://github.com/nicodv/kmodes.

4. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 12 (2/1/2011), 2825–2830.

5. Yan, Michael. Python Package for Gower Distance. https://github.com/wwwjk366/gower.

5 Appendix

5.1 Distance metrics

Descriptions of distance metrics we made use of in our investigation. Note that here x_i represents the ith feature of the datapoint x (in our case each datapoint represents a person).

5.1.1 Euclidean distance (for numerical data)

$\sqrt{\sum_i (x_i - yi)^2}$

5.1.2 Manhattan distance or L1 norm (for numerical data)

$\sum_i |x_i - yi|$

5.1.3 Canberra distance (for numerical data)

$\sum_i \frac{|x_i - y_i|}{|x_i| + |y_i|}$

5.1.4 Cosine distance (for numerical data)

$1 - \frac{x \cdot y}{\sqrt{x \cdot x}\sqrt{y \cdot y}}$

5.1.5 Jaccard distance (for categorical data)

$\frac{\text{Number of non-equal features}}{\text{Number of non-zero features}}$

5.1.6 Gower distance (for categorical and numerical data)

Makes use of Dice distance for categorical features and Manhattan distance for numerical features.

1. For each numerical feature i
   (a) Calculate a pairwise distance matrix S_i using the Manhattan distance metric.
   (b) Normalize the metric.
2. For each categorical feature i
   (a) One hot encode it to form multiple binary columns.
   (b) Calculate a pairwise distance matrix S_i using the Dice distance metric:
   Dice distance $= \frac{\text{NNEQ}}{\text{NNZ} + \text{NTT}}$
   Here NNEQ = number of non-equal columns, NNZ = number of nonzero columns and NTT = number of columns in which both values are 1.
3. Gower distance matrix $= \frac{\sum_i w_i \times S_i}{\sum_i w_i}$ where w_i is the weight of the i-th feature of the input.