Kavindra Sahabir
CSC 590

CSC Final Project

In our knowledge of the animal kingdom, every animal has their own way of mating. Peacocks have mating rituals that involve showing off their bodies, birds have elaborate songs that they sing, and humans have the recent invention of speed dating. A phenomenon that is a direct result of our desires to be as efficient as possible, speed dating involves a short 4 to 5 minute conversation between two participants who have the steep objective of sussing out whether or not they are interested in each other in the same amount of time it takes to get coffee from Starbucks. My interest is in seeing what kind of attributes lead to speed dating participants deciding that they've built up enough interest to go on another date. Therefore, I chose to perform data analysis and classification of a dataset containing data gathered from participants in speed dating events from 2002 to 2004. The dataset contained more than 100 attributes, some of which measured the participants' ratings of their dates based on six attributes: attractiveness, sincerity, intelligence, fun, ambition, and shared interests. Other attributes include demographic data like race and field of work.

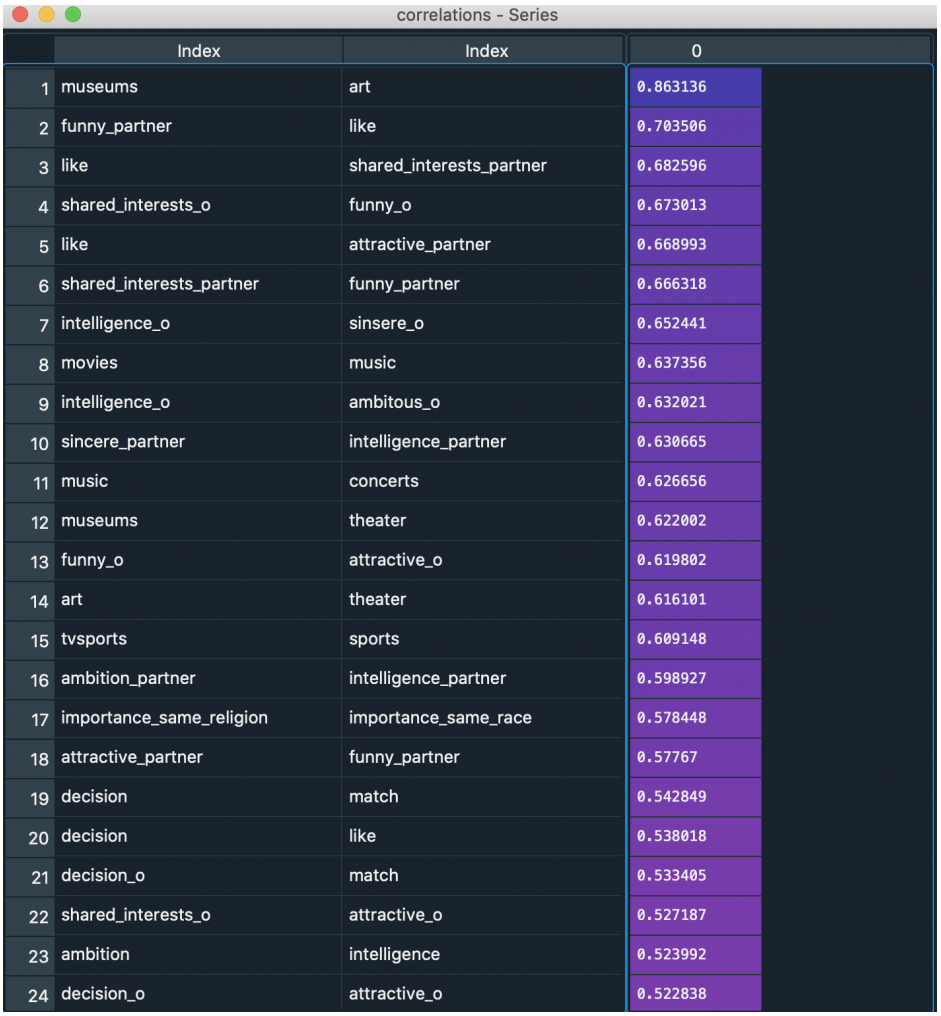
My analysis revolved around one question: what attributes could be used to predict whether or not a pair would be likely to have a second date?

The first step I took towards answering this question was to find the correlation between all the different attributes of the dataset. I found the Pearson correlation coefficient for each of the relationships between the attributes, and as an output, got a list of correlation coefficients as well as the attributes that have this correlation. Based on this list, I edited the DataFrame to include only these selected attributes. The attributes included are: the ages of the participants per date, the difference in ages of the participants per date, and the rankings of attractiveness, sincerity, intelligence, fun (humor), ambition, and shared interests that each participant in each date gave to each other (where attributes ending in _o represent rankings given to participant 2 from participant 1 and _partner are rankings given to participant 1 from participant 2). Also included are whether or not the participants per date were of the same race, and a measure of how much the interests of both participants were related (this attribute was included with the dataset and not calculated independently). These specific attributes are:
gender, age, age_o, d_age, samerace, match, shared_interests_o, shared_interests_partner, funny_o, funny_partner, attractive_o, attractive_partner, intelligence_partner, intelligence_o, sincere_partner, sinsere_o, ambition, ambitous_o, interests_correlate.

With the collection of all this data, there was bound to be missing values especially in the context of something as personal as dating. However, in order to analyze the dataset in the context of the initial question, I needed to have only fully complete data from as many dates as possible. I considered imputing the values, but decided that would be inappropriate as the data

in question is categorical rather than continuous. Imputation could have been performed by replacing missing values with the most common values per attribute. I decided against this in recognition of the uniqueness of the values that would come from each participant in each date. Therefore, I went ahead and dropped any rows with incomplete data. This left me with a dataset of 1048 rows, which I considered to be a big enough dataset to go ahead and analyze.

Fig. 1

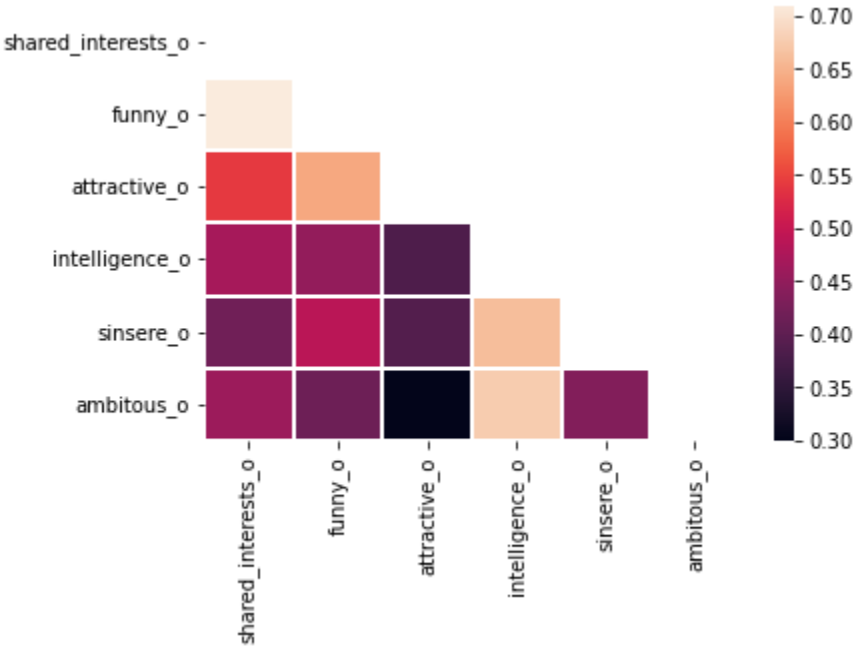


	Index	Index	0
1	museums	art	0.863136
2	funny_partner	like	0.703506
3	like	shared_interests_partner	0.682596
4	shared_interests_o	funny_o	0.673013
5	like	attractive_partner	0.668993
6	shared_interests_partner	funny_partner	0.666318
7	intelligence_o	sinsere_o	0.652441
8	movies	music	0.637356
9	intelligence_o	ambitous_o	0.632021
10	sincere_partner	intelligence_partner	0.630665
11	music	concerts	0.626656
12	museums	theater	0.622002
13	funny_o	attractive_o	0.619802
14	art	theater	0.616101
15	tvsports	sports	0.609148
16	ambition_partner	intelligence_partner	0.598927
17	importance_same_religion	importance_same_race	0.578448
18	attractive_partner	funny_partner	0.57767
19	decision	match	0.542849
20	decision	like	0.538018
21	decision_o	match	0.533405
22	shared_interests_o	attractive_o	0.527187
23	ambition	intelligence	0.523992
24	decision_o	attractive_o	0.522838

A sample of the list of correlation coefficients generated. The first two columns show the attributes associated with each correlation coefficient

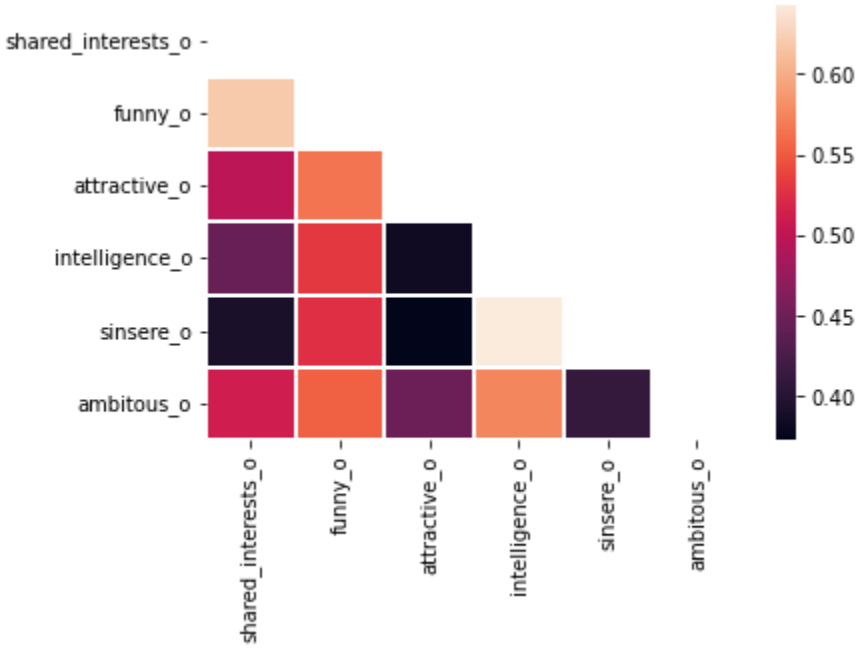
After finding this correlation data and editing the dataframe to “zoom in” on the relationships highlighted here, I wanted to analyze any differences in the correlations based on the gender of the initial participants. To this end, I created visual representations of the correlations to see what differences might exist.

Fig. 2



The visualization of the correlations between selected attributes for participants whose sex was specified to be males.

Fig. 3



A visualization of the correlations between selected attributes for participants whose sex was specified to be females.

As seen from the heatmaps, there were some attributes that seemed to be just as significant for males as they are for females. An example is the correlation between a participant's ranking in terms of funniness and their ranking in terms of how much that participant shares interests with their partner. There doesn't seem to be attributes that have significantly different correlations for the two sexes measured.

After these correlations were discovered, a Naive Bayes classifier was used on this dataset with the hopes of confirming that the attributes included in the edited dataframe are indeed good for predicting whether or not the participants would indicate a willingness for a second date. As hoped, the classifier resulted in fairly high accuracies around 85% with a maximum accuracy of 89.5%.

Fig. 4

df_norm - DataFrame

Index	gender	age	age_o	d_age	samerace	match	ıred_interest	ĵ_interests_p	funny_o	unny_partne	attractive_o	ractive_partr	lligence_part	ntelligence_	ncere_partn	sinsere_
0	0	0.176471	0.529412	0.428571	0	0	0.6	0.5	0.8	0.7	0.6	0.6	0.7	0.8	0.9	0.8
1	0	0.176471	0.235294	0.0714286	0	0	0.5	0.6	0.7	0.8	0.7	0.7	0.7	1	0.8	0.8
3	0	0.176471	0.294118	0.142857	0	1	0.8	0.8	0.8	0.7	0.7	0.7	0.8	0.9	0.6	0.8
4	0	0.176471	0.352941	0.214286	0	1	0.7	0.6	0.6	0.7	0.8	0.5	0.7	0.9	0.6	0.7
5	0	0.176471	0.411765	0.285714	0	0	0.7	0.4	0.8	0.4	0.7	0.4	0.7	0.8	0.9	0.7
6	0	0.176471	0.705882	0.642857	0	0	0.7	0.7	0.5	0.4	0.3	0.7	0.7	0.7	0.6	0.6
8	0	0.176471	0.588235	0.5	0	1	0.9	0.8	0.8	0.9	0.7	0.7	0.8	0.8	0.6	0.7
9	0	0.176471	0.352941	0.214286	0	0	0.6	0.8	0.6	0.8	0.6	0.5	0.6	0.6	0.6	0.6
10	0	0.352941	0.529412	0.214286	1	0	0.4	0.3	0.9	0.4	0.8	0.5	0.8	0.6	0.7	0.7
11	0	0.352941	0.235294	0.142857	1	0	0.5	0.6	0.6	0.6	0.7	0.8	0.6	1	0.5	0.6
12	0	0.352941	0.235294	0.142857	0	0	1	0.4	1	0.6	1	0.5	0.9	1	0.8	1
13	0	0.352941	0.294118	0.0714286	1	1	0.9	0.7	0.9	0.6	0.9	0.7	0.7	0.9	0.9	0.9
14	0	0.352941	0.352941	0	0	0	0.8	0.8	1	0.9	1	0.6	0.7	1	0.8	1
15	0	0.352941	0.411765	0.0714286	1	0	0.7	0.2	0.5	0.3	0.7	0.8	0.8	0.7	0.7	0.8
16	0	0.352941	0.705882	0.428571	1	0	0.3	0.9	0.3	0.6	0.5	0.7	0.9	0.4	0.5	0.3
17	0	0.352941	0.529412	0.214286	1	0	0.5	0.5	0.7	0.5	0.7	0.5	0.7	0.7	0.8	0.7

A sample of the edited dataframe, which was normalized per attribute and includes the attributes discussed above

We then used the .argmax function to select the highest probability and generate a list of predictions according to the class with the highest probability per row in the testing dataframe.

The results of this analysis shows some nicely insightful data that gives a good indication of what speed dating participants prioritize when considering their willingness to go on a second date. For one, the strongest correlation (that makes sense in the context of the attributes considered) is between the funninness of participant 2 and whether or not the participant 1 would say they like participant 2. Another interesting correlation is between how much participant 2 believes they share in terms of interests, and whether or not they like their partner. Another interesting insight is that one's intelligence and their perceived sincerity are also well correlated. Also included is the cliche that the attractiveness of a person and whether or not they are liked, which seems to be backed up by the correlation shown between those two attributes. It would seem that, even more than attractiveness, speed dating participants prize having shared interests and having partners who have a nice sense of humor. This signals a shift away from dated ideas in what humans look for in partners, although to be fair this data is collected from participants in a first world country. The results of this dataset's analysis may not hold up in different countries with different societies/different standards of qualities to look for in partners.

Appendix:

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn import preprocessing


df = pd.read_csv('~/Downloads/speeddating.csv', na_values = '?')

label = preprocessing.LabelEncoder()

df.dropna(inplace=True)

correlations = df.corr().unstack().sort_values(ascending=False).drop_duplicates()

#print(correlations)

#create correlation heatmaps based on gender

df1 = df[['gender','age','age_o','d_age','samerace','match',
        'shared_interests_o','shared_interests_partner','funny_o','funny_partner',
        'attractive_o','attractive_partner','intelligence_partner',
        'intelligence_o','sincere_partner','sinsere_o','ambition','ambitous_o',
        'interests_correlate']]

#create a list of attributes that show what the person thinks of the other person


att = ['shared_interests_o','funny_o','attractive_o','intelligence_o','sinsere_o','ambitous_o']

#heatmap for men
#np.triu is to only have the relevant data

df1_male = df1[df1['gender']=='male']
#df_ma = sns.heatmap(df1_male[att].corr(), mask = np.triu(df1_male[att].corr()),linewidths = 0.1,
linecolor='white')


#heatmap for women
df1_female = df1[df1['gender']=='female']
df_fem = sns.heatmap(df1_female[att].corr(), mask = np.triu(df1_female[att].corr()),linewidths =
0.1, linecolor='white')

df1['gender']= label.fit_transform(df['gender'])
```

```
#label value of 0 is female, value of 1 is male

def normalize_series(s):
    return (s - s.min()) / (s.max() - s.min())

df_norm = df1.apply(lambda x: normalize_series(x))

#using Naive Bayes classifier to predict matches

df_train = df_norm.sample(frac=2/5)
df_test = df_norm[~(df_norm.index.isin(df_train.index))]

df_train.head()

print(len(df_train), len(df_test), len(df1))

#function to find the number of instances for attribute per class

def find_counts(df, splitby):
    count = {}

    # determine values of dataset split per class
    value_counts = df[splitby].value_counts().sort_index()
    #convert to numpy array for elementwise calculations
    count["class_labels"] = value_counts.index.to_numpy()
    count["class_counts"] = value_counts.values

    # determine conditional probabilities for each attribute per class

    # for each element in each of the columns, excluding the Type column
    for i in df.drop(splitby, axis=1).columns:

        #create empty dictionaries to store the condiitonal probabilities for each attribute
        count[i] = {}


        # find number of values for each attribute for each class
        counts = df.groupby(splitby)[i].value_counts()
        #print(counts)

        #pivot dictionary so that the dictionary is split into groups based on columns,
        #and each column is each class
        training_counts = counts.unstack(splitby)
```

```
        # table contains NaN values after pivoting
        #(since not all values for each attribute exist in all the class labels)
        #therefore, we add a count of 1 to every value to solve this
        #this does not affect the final result as the ratios stay intact

        if training_counts.isna().any(axis=None):
          training_counts.fillna(value=0, inplace=True)
          training_counts += 1


        # calculate probabilities for each attribute belonging to each class
        df_probabilities = training_counts / training_counts.sum()

        #print (df_probabilities)
        #storing the probabilities in the empty dictionaries created with count[i]={}
        for j in df_probabilities.index:
            probabilities = df_probabilities.loc[j]
            count[i][j] = probabilities
            #print(probabilities)

    return count

find_counts = find_counts(df_norm,"match")

#function for prediction
def predict_example(row, dict):

    #assigning class counts arrays to variable class_estimates
    class_estimates = dict["class_counts"]

    #for each index in dictionaries created to store values in 'find_counts' function
    for i in row.index:

        # need to handle cases where values are only found in the test dataset
        # so skip those cases and don't take them into consideration
        #for final calculations
        try:
            value = row[i]
            probabilities = dict[i][value]
            class_estimates = class_estimates * probabilities
```

```
            except KeyError:
                continue

        #find the probability of each row's class label
        #by finding the maximum probability calculated
        #out of the three probabilities found for each row
        max_value = class_estimates.argmax()

        #calling the predicted class labels
        prediction = dict["class_labels"][max_value]

        return prediction

# applying the function to the test dataframe
#axis = 1 ensures that the function is applied to each row
# args takes in find_counts for the row variable
predictions = df_test.apply(predict_example, axis=1, args=(find_counts,))

#check accuracy by appending predictions to testing dataframe
#and checking how many rows are predicted correctly
df_test['predicted class label'] = predictions

accuracy = []
comparison_column = np.where(df_test["match"] == df_test["predicted class label"], True, False)
for i in comparison_column:
    if i == True:
        accuracy.append(1)
```

Technical explanation:

In this project, I did some exploratory data analysis as well as classification utilizing a Naive Bayes classifier that is specifically made for datasets with categorical variables.

The first thing I did was set na values in my dataset equal to the character “?”, as missing values in the dataset were represented by the “?” character in the original dataset. I also made use of the scikit.learn label encoding function to assign the string type variables as ordinal numerical values. I mainly used this for the gender attribute, as that was the only variable with

string type variables that I included in the dataframe constructed after finding the correlation of the columns in the original dataset. After editing the dataframe, I performed normalization on all the columns because some columns had rankins with a range of 0-10 and others had a range of 0-5. Therefore, each column was normalized to fit a 0-1 range. This normalization, which was min/max normalization, was performed by a function that was applied to each attribute and row with the use of a ".apply" function.

The Naive Bayes classifier used was specifically designed for working with categorical data. All that was needed was to find the general probabilities of a value belonging to a specific class in the specified dataset. This is referred to as the prior probability. This results in two values, each being a probability of a value belonging to either 0 or 1 in the "match" column (0 represented no second date and 1 representing going on a second date). Then, we needed to find the conditional probability of a value belonging to a specific attribute as well as a specific class. To achieve this, we counted the number of values that belonged to each of the attributes and divided them by the total number of values for each attribute per class. Finally, we multiplied the prior probabilities by the conditional probabilities to get a dictionary of lists with 2 values (the probability of each individual value belonging to each class). Then, the .argmax function was used to select the highest probability and generate a list of predictions according to the class with the highest probability per row in the testing dataframe. The accuracies ranged from 83% to 90% in the 10 times the classifier was run, which shows that the attributes selected in the beginning were good at predicting whether or not the participants would go out on a second date or not.

Acknowledgements:

This report could not have been completed without the help of Prof. Venkatesh. His efforts to ensure that I was able to complete my semester with a deep understanding of the material are greatly appreciated. Thank you, Professor!